

SoK: Hardware-Supported Cryptographic Protection of Random Access Memory

Roberto Avanzi^{1,2}, Andreas Sandberg³, Ionuț Mihalcea³,
David Schall⁴ and Héctor Montaner⁵

¹ Arm Germany GmbH, Grasbrunn, Germany

roberto.avanzi@arm.com

² Caesarea Rothschild Institute, University of Haifa, Israel

roberto.avanzi@gmail.com

³ Arm Limited, Cambridge, United Kingdom

{andreas.sandberg,ionut.mihalcea}@arm.com

⁴ School of Informatics, University of Edinburgh, United Kingdom

david.schall@ed.ac.uk

⁵ Graphcore, Cambridge, United Kingdom

hector.montaner@outlook.com

Abstract. Confidential Computing is the protection of data in use from access or modification by any unauthorized agent, including privileged software. For example, in Intel SGX (Client and Scalable versions) and TDX, AMD SEV, Arm CCA, and IBM Ultravisor this protection is implemented via access control policies. Some of these architectures also include memory protection schemes relying on *cryptology*, to protect against physical attacks. We review and classify such schemes, from academia and industry, according to protection levels corresponding of adversaries with varying capabilities, budget, and strategy.

The building blocks of all cryptographic memory protection schemes are encryption and integrity primitives and modes of operation, as well as anti-replay structures. We review the possible implementation of these building blocks, and discuss their relative merits. We consider their possible combinations of these alternatives, and evaluate the performance impact of the resulting schemes.

In order to be able to perform a relatively quick evaluation of hundreds of variations, we introduce a framework for performance evaluation in a simulated system based on the SimPoints methodology. To understand the best and worst case overhead, we also consider systems with varying load levels.

Finally, we propose new solutions to further reduce the performance and memory overheads of such technologies. Advanced counter compression techniques make it viable to store counters used for replay protection in a physically protected memory. By additionally repurposing some ECC bits to store integrity tags, we can provide the highest levels of confidentiality, integrity, and replay protection at a hitherto unattained performance penalty, namely 3.32%, even under extreme load and at costs that make them reasonable in data centers. Combinations of technologies that are suitable for client devices are also discussed.

Keywords: Security and privacy · Hardware-based security protocols · Memory Encryption · Memory Integrity · Lightweight ciphers · Integrity Trees

Contents

1	Introduction	3
2	Summary of the Results	4
3	Definitions and Background Material	8
3.1	Definitions	8
3.2	Adversaries	8
3.3	Protection Levels	9
3.3.1	L0: Access control	9
3.3.2	L1: Memory encryption	9
3.3.2.1	Implementation aspects.	10
3.3.3	L2: Encryption and integrity verification	10
3.3.3.1	MirE: <i>Message Authentication Codes</i> (MACs) in repurposed ECC bits.	10
3.3.3.2	Implementation aspects.	10
3.3.4	L3: Encryption, integrity, and replay protection	11
3.3.4.1	Implementation aspects.	11
4	Review of the Building Blocks	11
4.1	Cryptographic Primitives	11
4.1.1	Memory encryption primitives	11
4.1.2	Authentication primitives	12
4.1.3	Modes of operation	13
4.2	Memory integrity structures	13
4.2.1	Memory overhead comparison	15
4.2.2	Methods Excluded from Performance Evaluations	15
4.2.2.1	Log Hashes.	15
4.2.2.2	Morphable Counters.	15
4.2.2.3	Isolated Tree with Embedded Shared Parity.	16
4.2.2.4	Approximate Set Membership.	16
4.3	Cryptographic parameters and practices	17
4.4	On the design space	17
5	Benchmarking plan, results, and discussion	18
5.1	Benchmarking environment and methodology	18
5.2	Selection of the benchmarking sets	19
5.2.1	Simulation of system load	20
5.2.2	Baseline performance	20
5.2.3	Initialization of short counters	20
5.3	Set 1: State-of-the-Art, AES vs. Lightweight encryption ciphers, and 64-bit vs. 32-bit MACs	21
5.4	Set 2: Impact of MPE cache sizes	23
5.5	Set 3: Impact of the cache line length	23
5.6	Set 4: Asynchronous MAC verification	24
5.7	Set 5: Use of on-chip memory for L2 and L3	24
5.8	Set 6: Impact of repurposing ECC Bits, 3-way split counters, and large counter caches	24
5.9	Set 7: Impact of incremental MACs	26
5.10	Set 8: Breakdown of selected configurations	26
5.11	Set 9: Impact of <i>Read-Modify-Write</i> (RMW) operations	29
5.12	Remarks on area and power	29
6	Conclusions and Open Questions	30
	References	31

1 Introduction

Cloud computing promises to increase efficiency and drive down cost for users. Such services co-locate multiple mutually untrusted tenants in the same data center and sometimes even on the same physical machines. Compared to traditional on-premises solutions, users of cloud computing face two additional threats. First, hostile tenants may try to exploit bugs in the hypervisor or *access control* mechanisms to impact the confidentiality, integrity, or availability of co-located virtual machines. Second, compromised insiders at the service provider or its contractors may try to gain access to customer data.

Similar threats exist in client devices, such as phones, which have evolved into smart terminals and identity providers. Like in a data center, adversaries may use co-located untrusted code or even have physical access to the device. Use cases such as secure payments, secure identification, and software anti-piracy rely on strong confidentiality and integrity guarantees. These are often provided in separate components, e.g., SIM cards, USB tokens, or *Trusted Platform Modules* (TPMs). Consolidating their functionality onto the main *System-on-a-Chip* (SoC) enables new use cases while reducing total costs — but also enables opportunities for the aforementioned adversaries.

AMD SEV [KPW16], Arm CCA [MPS⁺21], Intel’s Client SGX [Gue16a] and Scalable SGX [JMSS20], and Intel TDX [Int21] move towards this goal by providing access control mechanisms. The latter are managed by a HW-supported *Trusted Computing Base* (TCB). Some of these technologies even include protection against adversaries with physical access to the system. For instance, Intel’s Client SGX implements a *Memory Encryption Engine* (MEE) [Gue16a] that provides confidentiality, as well as integrity and protection against replay attacks. Such strong security guarantees can be very costly in terms of performance and storage. For this reason, AMD SEV, Intel TDX, and Scalable SGX (the latter two sharing the same memory protection scheme) provide weaker guarantees in exchange for better performance.

In this study we review what cryptographic technologies are available to protect the contents of data-in-use in RAM against an adversary with physical access to the system. We also evaluate the costs associated to the most meaningful combinations of these technologies.

The starting point is a thorough review of the techniques documented in the scientific and technical literature. Even though we cite several architectures for implementing complete *Trusted Execution Environments* (TEEs), the scope of this paper does not address aspects such as *Operating System* (OS) and Hypervisor support, I/O, virtualization, attestation and IPC mechanisms. We focus on solutions for cryptographic memory protection that are entirely implemented within the SoC package limits.

For real-world applications, understanding the cost of a solution is crucial. Area and power constraints limit the viable options, but relaxing them can be justified by strong market requirements. On the other hand, solutions with high performance penalties and memory overheads risk being rejected without further consideration of their merits. For this reason, we compare the costs of several schemes and variations thereof, where *we focus mainly on performance penalty and memory overheads*. We also propose new methods to further reduce these costs.

Our performance evaluation uses the entire industry-standard SPEC 2017 [BLvK18] benchmark suite running on the `gem5` simulator [BBB⁺11, LAA⁺20].

This work also fills a gap in the literature, as there are only very few papers surveying the subject. The 2009 paper [ECG⁺09] is a survey of memory integrity schemes, intended as *full integrity*, i.e., including replay protection. The 2013 paper [HT13] contains a thorough survey of memory encryption techniques until its publication, but its performance data is taken from the surveyed papers, which more often than not cannot be properly compared to each other. Its abstract states “*To date, little practical experimentation has been*

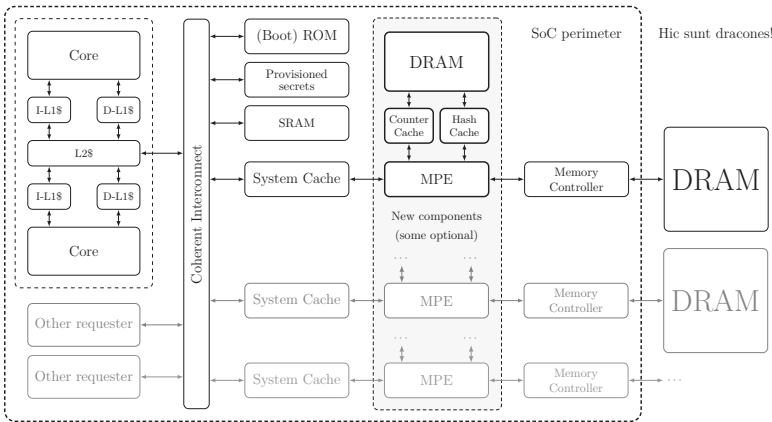


Figure 1: Simplified system level view of a SoC with Memory Protection Engine(s).

conducted, and the improvements in security and associated performance degradation has yet to be quantified.” Ten years later, this sentence still holds true. The more recent papers [Shw15, MZLS18, SNR⁺18a, SNR⁺18b, TSB18, TBC⁺20, SNOT21, IMO⁺22] (and many more) only compare their proposals to few other schemes. We conjecture that one reason for the limited comparisons is the long-running time of simulations, a serious issue we address with the use of SimPoints (cf. Section 5.1). The recent survey on HW-supported TEEs [SMS⁺22] focuses on the offered features and does not address performance penalties and memory overheads: We refer the reader to it for information on overall Architecture, supported types of isolated processes, Attestation, dynamic/static nature of the *Root of Trust* (RoT), support for secure I/O and storage, and size of the TCB.

Outline of the paper. We open with a summary of the contributions in Section 2. Section 3 contains background material, such as: the models of the adversaries and a discussion of the memory protection levels. Section 4 contains a review of the cryptographic primitives and memory integrity structures; as well as a discussion of cryptographic parameters such as key and MAC lengths. Section 5 describes the actual benchmarks and discusses how these support the claims in Section 2. In Section 6 we conclude.

2 Summary of the Results

Cryptographic memory protection relies on:

- (i) Encryption,
- (ii) Authentication, and
- (iii) Replay protection structures.

In the third group we also include the protection of *a relatively small amount* of RAM, such as placing it on-chip, or in a tamper-proof or -evident device with a secured communication channel to the SoC. We exclude applying this approach to the entire RAM, because cost and thermal considerations would make it impractical for general use.

There are only a few meaningful combinations of these technologies, forming various *Protection Levels*, the first one comprising only access control, and three increasingly robust level based on cryptography, which may or may be not combined with the first:

L0: Access Control only;

Table 1: Selected documented TEEs and cryptographic memory protection schemes.

Scheme or Technology	Year	Level	Encryption	Authentication	Integrity structure (and other technologies)	References
Hall and Jutla’s PAT	2002	—	—	—	Counter Tree	[HJ08]
AEGIS	2003	L3	AES-CBC	Incremental hash	<i>Merkle Tree</i> (MT)	[SCG ⁺ 03a, GSC ⁺ 03]
AEGIS (alt. design)	2003	L3	“OTP”	MD5/SHA-1	Log Hashes	[SCG ⁺ 03b]
Yan et al.	2006	L3	AES-GCM	GMAC	<i>Split Counter Tree</i> (SCT)	[YEP ⁺ 06]
SecureMe	2007	L3	AES-CTR (AISE)	SHA-1/HMAC	Bonsai MT	[RCPS07]
Flicker, resp. SEA	2008	L0	—	—	(TCB-managed <i>Page Tables</i> (PTs))	[MPP ⁺ 08a], resp. [MPP ⁺ 08b]
Bastion	2010	L0 + L3	AES-ECB	AES-CMAC	MT	[CL10]
IVEC	2010	—	—	GMAC	SCT	[HS10]
SecureBlue++	2011	L3	Undocumented	Undocumented	“Integrity tree”	[WB11]
H-SVM, resp. SICE	2011	L0	—	—	(TCB-managed PTs)	[JACH11], resp. [ANZ11]
Hyperwall	2011	L0	—	—	(TCB-verified PTs)	[SL12]
HyperCoffer	2013	L3	(Same as SecureMe)	(Same as SecureMe)	(Same as SecureMe)	[XLC13]
Intel’s Client SGX1/SGX2	2013	L0 + L3	AES-CTR	Encrypted UHF	{ Counter Tree only on a small <i>Enclave Page Cache</i> (EPC), and secure paging from EPC to other RAM	[MAB ⁺ 13]
Iso-X, resp. EqualVisor	2014	L0	—	Optional	Optional	[EEO ⁺ 14], resp. [DZWL14]
PodArch	2015	L0 + L3	AES-GCM	GMAC	On-demand secure encryption	[Shw15]
AMD-SEV {-SNP}	2016	L1	AES-XEX	—	(TCB-managed Nested PTs in SNP)	[KPW16, AMD20]
VAULT	2018	L3	Unspecified	Unspecified	Variable arity tree, encrypted MAC-less leaves	[TSB18]
Delta-Encoding	2018	L3	AES-CTR	Encrypted UHF	Delta Encoded SCT	[YA18]
Morphable Counters	2018	L3	AES-CTR	Encrypted UHF	Morphable Counters SCT	[SNR ⁺ 18a]
SYNERGY	2018	L3	AES-GCM	GMAC	Bonsai MT, MACs in ECC bits	[SNR ⁺ 18b]
TIMBER-V	2019	L0	—	—	(Tagged Memory)	[WWB ⁺ 19]
Apple’s Secure Enclave	2020	L0 + L3	AES-XEX	AES-based CMAC	Bonsai “Integrity tree”	[App20]
Intel TDX, Scalable SGX	2020	L0 + L2	AES-XEX	Reduced SHA-3	MACs in ECC bits	[JMSS20, Int21]
Keystone	2020	L0 ++	AES-128, unspecified mode	Unspecified	Secure on-chip ↔ external RAM paging	[LKS ⁺ 20]
Arm CCA	2021	L0	—	—	(TCB-managed page ownership table)	[MPS ⁺ 21]
IBM Ultravisor	2021	L0	None on Power9	None on Power9	(Tagged memory addresses)	[HPL ⁺ 21]
PENGLAI	2021	L0 + L3	Unspecified	Unspecified	Dynamically allocated MT	[FLD ⁺ 21]
ELM	2022	L0 + L3	Flat-OCB (OCB)	Flat-OCB and PXOR-MAC	Counter Tree	[IMO ⁺ 22]
CSI:RowHammer	2023	—	Optional	PMAC	MACs in ECC bits	[JLK ⁺ 23]

L1: Memory encryption;

L2: Memory encryption and integrity; and

L3: Memory encryption, integrity, and protection against replay attacks.

While one can imagine use cases for various degrees of integrity protection only without encryption, we are not aware of any such scheme.

We implement Protection Levels L1 to L3 in the *Memory Protection Engine* (MPE), an IP block sometimes known as *Memory Encryption Engine* (MEE), e.g., in SGX. As depicted in Fig. 1, in a typical SoC the MPE sits between the main interconnect (or a system cache) and a memory controller. It can optionally have its own caches, and even access to a physically secure private DRAM to store metadata.

As a starting point for choosing the components used to implement each Protection Level, we first review the state-of-the-art. Table 1 outlines cryptographic memory protection in various TEEs. While the TEE list is not exhaustive (a more complete list is given in [SMS⁺22]), the list of primitives and structures is comprehensive, except for some deprecated methods and some exclusions as explained in Section 4.2.2. (These technologies are detailed in Sections 4.1 and 4.2.)

We obtain the following groups of alternatives:

1. *The AES vs. a lightweight cipher suitable for memory encryption.* The selected lightweight cipher is QARMA-128 [Ava17], as explained in Section 4.1.1.
2. *Direct encryption*, where a plaintext block is input to the cipher to compute the corresponding ciphertext, *vs.* Counter mode (CTR) *encryption*, where the encryption of successive counter values results in a *keystream* which is then XOR-ed to the plaintext to obtain the ciphertext (cf. Section 4.1.3 for more details).
3. *Various MAC algorithms for memory integrity*, such as Carter-Wegman *Universal Hash Functions* (UHF) [CW79] (for instance, encrypted linear functions of the message), encrypted checksums of the plaintext, or *Parallel MAC* (PMAC) [Rog04] (see Section 4.1.2 for a discussion of the options).
4. *The choice of 32-bit vs. 64-bit MACs for the integrity tags.*
5. *Different sizes of the caches used by the MPE, as well as on-chip memory to store MACs or counters.*
6. *Optionally repurposing some ECC bits to store MACs.*
7. *Different sizes of the memory regions protected by one MAC.* This is obtained both varying the CL size and letting a single MACs cover multiple CLs.
8. *Synchronous vs. asynchronous integrity verification.*
9. *Integrity counter trees with increasing arity.* Their nodes, which in this paper always fit in one CL, contain from 8 to 256 highly compressed counters, one for each child. To achieve such high arities, the operations on the tree are designed to guarantee that the most significant bits (e.g., 56 bits) of all counters in a node are equal. This common part is stored once in the node, and the least significant bits of each counter are stored individually (cf. Section 4.2.) We also study counters split into *three parts*.

We simulate various combinations of the above alternatives in the `gem5` simulator, and run the benchmark suite in these simulated systems with different loads on the memory subsystem. To our knowledge, this is the first evaluation of this type. We also randomize the internal state of the system structures to simulate the more realistic performance characteristics of a not-freshly booted system.

The main two results are:

- R1** Nearly-transparent strong memory protection is possible with current technology, for both client and server systems and in most conditions (cf. Section 5.3 for L1 and L2, and Sections 5.8 and 5.11 for L3. See also Section 6).
- R2** Lightweight ciphers clearly outperform the AES in area and power (cf. Section 5.12), but their impact on performance is major only in L1 and L2 schemes (cf. Sections 5.3, 5.7 and 5.8). For L3 schemes that use counter based encryption, the primary influence on system performance is the organization of the data integrity structures, and the choice of cipher is secondary.

More detailed results and other observations follow:

- R3** The performance of schemes based on *direct encryption* methods, such as L1 and L2 schemes, is very sensitive to the latency of the cipher. Moving from the AES to QARMA brings a significant reduction in performance loss. (Cf. Section 5.3.)
- R4** Regarding the previous result, the performance penalty depends much more on the additional decryption latency on memory reads than on the additional latency induced by encryption on memory writes. Also, linear interpolation based on the encryption cipher's latency provides a good approximation of the performance penalties, with our measurements and cipher latency smaller than the AES-256's. (Cf. Section 5.10.)
- R5** Using 32-bit in place of 64-bit MACs halves MAC memory requirements, which is significant. However, MAC memory accesses have poor spatial locality, and the impact on performance is marginal (cf. Section 5.3).
- R6** Similarly, the MAC caches are not a major performance factor. Counter caches are more effective. The relative improvements due to caching increase with the load of the system. (Cf. Section 5.4.)
- R7** Using longer CLs (i.e., 128 B instead of 64 B) does not necessarily improve overall performance significantly. However, it halves the memory used by the MACs and enables more aggressive metadata packing in the counter trees. (Cf. Section 5.5.)
- R8** While asynchronous integrity verification improves performance, it is a security risk as the system may speculate on potentially corrupted data (cf. Section 5.6).
- R9** If MACs [are stored] in repurposed ECC bits (short: MirE) or in an internal memory, the L2 and L3 performance has a major improvement (cf. Sections 5.7 and 5.8).
- R10** We find that incremental MACs covering multiple CLs, while significantly decreasing the memory overhead, adversely impact performance. The degradation is in part due to the fact that we do not implement an optimization that exploits the compressibility of the plaintext to reduce memory accesses. This optimization may compromise plaintext confidentiality, hence it represents a severe security risk (cf. Section 5.9).
- R11** Increasingly higher arity counter trees offer major and progressive reduction in both memory overhead (cf. Table 2) and performance penalties. However, as the arity of such integrity trees increases, with the counter group size staying constant, the system must re-encrypt memory or regenerate integrity nodes increasingly often. The use of 3-way split counters substantially reduces the cost of these RMWs operations. (Cf. Sections 5.3, 5.8 and 5.11.) The resulting scheme is more effective than techniques such as Morphable Counters, which are very complex to implement (cf. § 4.2.2.2).
- R12** In this paper for the first time we show the advantages of counters split into *three parts*. The leaf level of our 3-way, high arity split counter trees is just 1:128 or 1:256 of the total RAM. It is small enough to be stored in a physically secure, on-chip or in-package memory. No other tree levels need to be maintained. This enables L3 schemes with very good performance. Combined with MirE, the performance penalty under extreme bus contention is just 3.32%. (Cf. Sections 5.7, 5.8, 5.10 and 5.11.)

3 Definitions and Background Material

3.1 Definitions

Following the Arm terminology [MPS⁺21], a *Realm* is a process domain that is isolated from other process domains through policies enforced by a small TCB. This term encompasses both small *Enclaves* as well as processes and larger virtual machines.

The SW-accessible volatile, external memory, connected to a memory controller, is seen as an array of blocks. These blocks match the *Last Level Cache* (LLC)'s *Cache Line* (CL) size and are thus also called CLs.

An encryption or authentication function is said to provide *spatial uniqueness* if, when computed on equal inputs, but written to different locations, it results in different outputs. This is achieved by including the *Physical Address* (PA) of the encrypted or authenticated CL in the computation.

An encryption or authentication function provides *temporal uniqueness* (*freshness*) when repeated writes of the same plaintext to the same location result in different outputs. This is achieved by including a counter in its computation.

In what follows a mode (of operation) is a general purpose encryption mode of operation. A *Memory Encryption* (ME) mode is understood to be an encryption mode of operation with plaintext and ciphertext having the size as a CL, and no associated data.

An *on-chip* component is defined as a physically secure block in the same package as the processing elements. In this case the package shall be *tamper-averting*, i.e., a package that is either tamper-proof/resistant, or tamper-evident/detecting.

3.2 Adversaries

To adequately answer the question posed in the Introduction, we categorize technologies based on the considered adversaries. The latter are distinguished according to their access to the target, and their resourcefulness. Before doing this, however, we must make a few critical remarks. Cryptographic memory protection cannot completely address most side channels, including those that exploit physical effects, hence these are out of scope. The exclusion applies to the access-pattern side channel as well: Adversaries can reverse engineer software properties or elicit secrets from access patterns. The only generic and provably effective mitigation would be *Oblivious RAMs* (ORAM) [Gol87], which carry prohibitive performance penalties. The same applies to SW exploitation, timing attacks and micro-architectural side-channels. For all these threats, mitigations should be applied to SW as needed. User-space services can always deny resources to Realms, including scheduled time, hence Denial-of-Service attacks cannot be completely mitigated. In particular, availability of memory cannot be guaranteed by cryptographic means once an adversary has access to the HW.

We can now define the following Adversaries:

- \mathcal{A}^{SW} can run SW on the target, and provide inputs to it, including through external interfaces.
- $\mathcal{A}_{\text{passive}}^{\text{HW}}$ has physical access to the system that contains the target, including its internals, but does not have the capabilities to access on-chip communication interfaces. They can interpose chips and modules for the sole purpose of monitoring transactions.
- $\mathcal{A}_{\text{active}}^{\text{HW}}$, also performs *active* attacks, e.g., blocking, corrupting, replaying or injecting transactions on the memory bus [KLR⁺20] or other interfaces.
- $\mathcal{A}_{\text{invasive}}^{\text{HW}}$ can mount highly invasive attacks at the chip or package level. Examples range from micro-probing attacks [Sko17] to actual chip reverse engineering and

editing using a Focused Ion Beam Microscope [TJ09]. $\mathcal{A}_{\text{invasive}}^{\text{HW}}$ is out of scope in this paper as the proper defenses require HW countermeasures.

SW and HW-capable adversaries are independent. The HW adversaries form a hierarchy $\mathcal{A}_{\text{passive}}^{\text{HW}} \subsetneq \mathcal{A}_{\text{active}}^{\text{HW}} \subsetneq \mathcal{A}_{\text{invasive}}^{\text{HW}}$.

3.3 Protection Levels

We provide detailed definitions of the Protection Levels. Table 1 shows how some documented solutions map to them. The technologies used to implement each level are listed. They are taken from options described in Section 2, Table 1. For more details about these technologies, cf. Section 4.1.

3.3.1 L0: Access control

Access control policies to implement *reverse sandboxing* are the first line of defense against \mathcal{A}^{SW} . However, RowHammer attacks (and micro-architectural side channels) have significantly increased the power of \mathcal{A}^{SW} , enabling them to bypass reverse sandboxing.

Physically separating memory rows of different process domains through access control and precise memory allocation policies could theoretically prevent RowHammer attacks. However, this approach requires complex system software changes and is impractical in real-world scenarios.

We do not discuss the implementation of L0.

From here on, we assume that appropriate access control policies are in place to stop unauthorized agents within the SoC, but not to prevent RowHammer attacks.

3.3.2 L1: Memory encryption

This level provides spatial uniqueness, but not temporal uniqueness.

Interest in L1 is driven by confidentiality requirements and to make attacks that depend on memory corruption (for instance RowHammer) more difficult. For this reason, L1 must use direct encryption with a cipher that enjoys a strong diffusion property, i.e., any input change induces a flip of each output bit with likelihood 1/2.

In general, protection against \mathcal{A}^{SW} is very limited, as is against $\mathcal{A}_{\text{passive}}^{\text{HW}}$ since the latter can detect ciphertext repeats. Also, note that attacks on the integrity of a system may still cause SW to reveal its contents, therefore this scheme alone does not guarantee confidentiality. Only full replay protection (L3) thwarts the particular attack just mentioned. Warm-boot and cold-boot attacks [HSH⁺09] are properly mitigated. Note that the same arguments apply also to L2.

A common requirement for L1 (and L2) system is the *cryptographic separation of Realms*, which serves to thwart combined SW/HW attacks based on the replay of memory from a target Realm into an adversary-controlled one. This can be achieved by per-Realm unique encryption *differentiators*. (Replay attacks into the *same* Realm, to reset it to a previously known state, require L3 protection.) The differentiators can be encryption keys or, if a single global encryption key is used, bit-strings to be used in a designated bit-field of the tweaks. Differentiators must be discarded upon Realm termination. They should not repeat. If they are tweak contributions, they can be implemented by, say, a TCB-managed 64-bit counter.

Address scrambling (a very lightweight encryption mechanism of the PA to permute the memory layout) may also be somewhat effective against RowHammer. It is deployed in some devices like smart cards for the purpose of mitigating side channel attacks. Note that since these schemes are usually static per boot session, address reuse can be detected:

this is often all an adversary needs to mount an attack. Hence, it should be considered only as an additional *defense-in-depth* measure and not as a complete mitigation per se.

3.3.2.1 Implementation aspects. With the AES, a CL is encrypted in *XOR, Encrypt, and XOR* (XEX) mode [Rog04], as in AMD SEV, TDX, and Apple’s Secure Enclave.

The chosen low-latency block cipher for memory encryption is QARMA-128 (as explained in Section 4.1.1). QARMA-128 is used in a Tweaked *Electronic Codebook* (ECB) mode as in Fig. 2a. In both cases the tweak is the address.

3.3.3 L2: Encryption and integrity verification

This level extends L1 with integrity tags, to detect memory corruption. It does not provide any temporal uniqueness, hence it must rely on a direct encryption method. An integrity tag is usually a MAC. Adversaries can still mount replay attacks.

L2 targets $\mathcal{A}_{\text{passive}}^{\text{HW}}$. It is also partly effective against $\mathcal{A}_{\text{active}}^{\text{HW}}$, if they only corrupt individual memory locations or have a limited time budget. To defeat targeted replay of memory including its integrity tags, stronger countermeasures are required (see Level L3).

This distinction within $\mathcal{A}_{\text{active}}^{\text{HW}}$, though seemingly arbitrary, is necessary due to varying complexities and costs not only of the attacks but also of the *countermeasures*. System designers can assess threats and make business decisions about accepting specific risks. Similarly, active Adversaries might opt for keeping their attacks passive at least initially, to avoid detection and to collect data for cryptanalysis.

3.3.3.1 MirE: MACs in repurposed ECC bits. If ECC memory is available, storing the MACs in (possibly some of) the ECC bits eliminates the need to reserve a RAM carve-out for the MACs, and significantly reduces memory traffic. Note that MACs are still accessible to \mathcal{A}^{HW} .

The Intel TDX MKTMEi is such a solution. We found no documentation on error correction in a TDX system, but the 28-bit MAC field size suggests that a *Single-Error Correction and Double-Error Detection* (SECCDED) (255, 247) Hamming code is used. This code is truncated to (143, 135) to cover 128 bits and 7 bits of the MAC each. The remaining 4 bits of the effective 576 bits in each CL are used for parity.

A different configuration is presented in [YA18], where a 56-bit MAC is used for integrity and error correction and a 7-bit ECC provides SECCDED to the MAC.

Remark 1. MirE raises the question of the performance impact of using ECC memory. Reported penalties are smaller than 0.5% [Bac14], stemming from increased traffic and additional processing in the DRAM controller: the impact is so small that we do not evaluate it as a separate configuration.

Remark 2. Beside SECCDED codes for 9-chip RAMs, there are several memory-specific *Reliability, Availability, and Serviceability* (RAS) designs, with varying levels of redundancy, starting with Chipkill [IBM99]. These are capable of handling also multiple errors. MirE can be easily implemented in these systems using suitable codes.

3.3.3.2 Implementation aspects. The same encryption techniques are used as for L1. For Intel TDX the MAC is computed using truncated SHA-3, with the latency assumed to be comparable to AES-128. In any other MirE scheme, following [JLK⁺23], the tag is computed using QARMA_{5-64- σ_0} . *Note that not all the ECC bits need to be repurposed for a MAC: these bits may contain both a shorter ECC and a MAC.* If the MACs are not stored in repurposed ECC bits, hashing is done by a *multilinear* UHF [CW79] at 32 or 64 bits. Note that these MACs are actually kept as unencrypted *hashes* while on-chip, which speeds up verification, and we encrypt them block-wise when they are evicted from

the hash cache groups. For instance, four 32-bit hashes are encrypted as a single 128-bit block. This enhances system robustness and security against corruption and replay attacks. In schemes with freshness (i.e., L3), the freshness data of the hashes that are encrypted together must be joined to form the common tweak for the hash block encryption.

3.3.4 L3: Encryption, integrity, and replay protection

This level is capable of detecting not only memory corruption, but also replay of memory contents including associated metadata.

With respect to L2, this level fully mitigates also against $\mathcal{A}_{\text{active}}^{\text{HW}}$. In order to replay a CL together with its counter and MACs the adversary either must successfully perform cryptanalysis or wait for a counter repeat. Note that in some variants, the counters themselves may be hidden to the adversary. More information about these data structures is found in Section 4.2.

Remark 3. In a L3 system, a single system-wide key is sufficient for authentication, since nodes closer to the root need to cover memory across Realms. In any case, this is not a security issue. Encryption differentiators are also not required, but they may be a hard customer requirement. Computing integrity tags on the ciphertext ensures that orphaned memory can still be verified, which is essential for secure erasure.

3.3.4.1 Implementation aspects. The same freshness information is included in the encryption and in the tag computation. A *CounTeR mode* (CTR) encryption mode is used with both AES (following AEGIS, the method by Yan et al., and SGX) and QARMA, except with *Encryption for Large Memory* (ELM), which uses Flat-OCB. The anti-replay technologies are described in Section 4.2.

4 Review of the Building Blocks

4.1 Cryptographic Primitives

4.1.1 Memory encryption primitives

RAM is commonly encrypted using a block cipher: the long initial latency of stream ciphers makes them unsuitable for the purpose.

For simplicity, we only consider block ciphers with a block size of 128 bits: smaller block sizes are used only for smart cards and small embedded devices, and longer blocks are uncommon. The selected block ciphers are the AES [DR02] and QARMA [Ava17], where the second is chosen as a representative of lightweight ciphers.

QARMA is a *Tweakable Block Cipher* (TBC): Beside the secret key and a text, a TBC accepts a *third* input known as a *tweak*, which is used together with the key to select the permutation computed by the cipher. Unlike the key, the tweak may be controlled by an adversary. TBCs simplify the design of modes of operation, with an early application to memory encryption [HT13].

The latencies of most suitable lightweight ciphers are similar (e.g., PRINCE [BCG⁺12]) or worse (for instance SKINNY [BJK⁺16]). To estimate performance penalties for these ciphers, readers can interpolate between our AES and QARMA results. A revised version of QARMA, QARMAv2 [ABD⁺23], has been introduced. Its latency is nearly equal to QARMA's, so we do not consider it as a separate configuration option.

Beside the AES, we do not consider other non tweakable block ciphers. The reason is that as they would require constructions that lead to increased latency anyway. We also do not consider ciphers with block sizes that make them less suitable for memory encryption: For instance SPEEDY [LMMR21] has a block size of 192 bits, and ASCON [DEMS21] in a

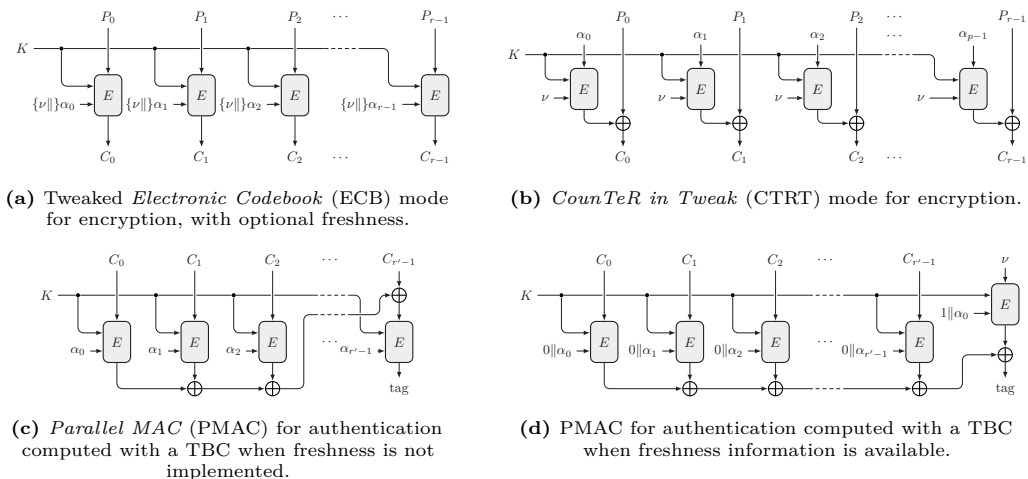


Figure 2: Encryption and authentication methods designed around a *Tweakable Block Cipher* (TBC). They show how freshness can lead to shorter critical paths. Notation: E is a TBC, the two inputs on the left side of the block being the key (above) and the tweak; $P = P_0\|\dots\|P_{r-1}$, resp. $C = C_0\|\dots\|C_{r-1}$ is the partition of a plaintext, resp. ciphertext in blocks of equal size; α_i is the *Physical Address* (PA) of the i -th block; and ν a nonce. If freshness is available, both encryption and authentication algorithms use it, and they share the same nonce. The TBC used for authentication may have a smaller block size than the encryption TBC, in which case $r \neq r'$.

tweaked mode such as *Masked Even-Mansour* (MEM) [GJMN16] has a block size of 320 bits. (For completeness' sake, given a public permutation $\pi : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$, we describe an example of a MEM construction: Given a key K , a tweak T , and a plaintext P , all n bits long, the ciphertext C is computed as $C = M \oplus \pi(P \oplus M)$, where $M = K \oplus \pi(T \oplus K)$.)

4.1.2 Authentication primitives

Standard hash functions such as SHA-2 [NIS12] or SHA-3 [NIS15] can be turned into MACs, but the resulting schemes are very slow and not parallelizable.

Carter-Wegman Hashes [CW79], i.e., encrypted UHF, are a better choice. UHF admit fully parallelizable constructions, such as multilinear functions of the input computed over a binary Galois field, as used in SGX [Gue16b]. If there is a MAC cache, it is actually the *not-yet-encrypted* UHF values that are cached, which are thus verified more efficiently.

Apple's Secure Enclave [App20] uses a *Cipher-based MAC* (CMAC) [IK03] to compute integrity tags. CMAC, being a block-wise chained construction, can not be made parallel and has a high latency, but Apple's use case does not need very high throughput. It is however unsuitable for general usage requiring high bandwidth and low latency. Instead, we evaluate TBC-based PMACs [Rog04]. PMACs are more expensive than encrypted UHF, but they can be used for error detection and correction beside integrity, cf. [HS10, SNR⁺18b, JLK⁺23]. The computation of PMACs is depicted in Figs. 2c and 2d. Such constructions can easily be made *incremental* where, upon a write, only the part of the message that has changed needs to be recomputed. A variant for non-TBCs, called PXOR-MAC is described in [IMO⁺22].

Encrypted checksums of the plaintext as in Rogaway's *Offset Codebook mode* (OCB) mode [Rog04] are an inexpensive method to compute integrity tags, but they suffer from two drawbacks. First, they need to be verified after decryption, potentially worsening overall latency. Second, since they require freshness, a CTR encryption should be used which has lower latency than direct encryption. With CTR encryption, using checksums of the plaintext as the basis for integrity would make the ciphertext malleable, whence a UHF-based MACs should be used instead.

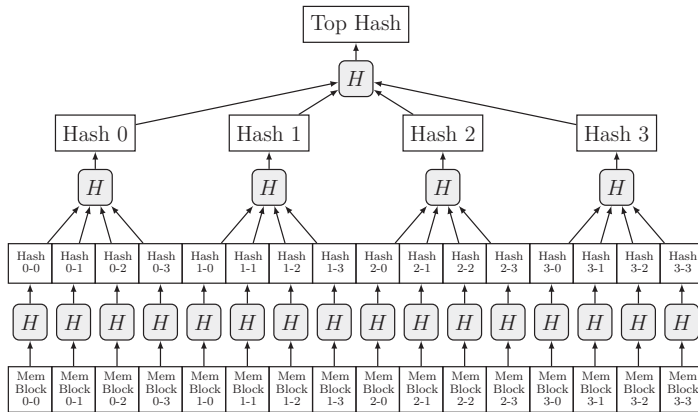


Figure 3: Merkle tree. H is a hash function.

4.1.3 Modes of operation

For memory encryption, many (authenticated encryption) modes of operation can be simplified somewhat because the length of the payload is a fixed multiple of the underlying cipher’s block length.

Some older schemes, such as Bastion [CL10], use the block cipher in *Electronic Codebook* (ECB) mode, but the lack of spatial uniqueness keeps plaintext patterns in the ciphertext, therefore modes that provide spatial uniqueness are necessary.

For direct encryption, spatial uniqueness is achieved by using the PA as the tweak. With a non tweakable block cipher, the latter is used in the *XOR, Encrypt, and XOR* (XEX) construction [Rog04], which is just the XTS mode of operation [IEE19] for a message whose length is a multiple of the block size. XEX is defined as $C_i = E_K(P_i \oplus M_i) \oplus M_i$. In other words, a tweak-derived *mask* is added to the input and the output of the cipher. The first mask M_0 is derived by encrypting the tweak, and the successive masks M_i for $i \geq 1$ are obtained by multiplying the first mask by a fixed sequence of values. Using a single finite field element γ we can put $M_i = \gamma^i \cdot M_0$. Inoue et al. introduce a Flat-OCB mode [IMO⁺22] which is similar to OCB [Rog04]. They define the L3 scheme ELM using Flat-OCB mode for data and PXOR-MAC to authenticate counter groups.

With a TBC, the PA (concatenated with freshness if provided) of each block is used directly as a tweak, cf. Fig. 2a, and a XEX construction is not needed.

In CTR encryption with a TBC, the counter and PA are used as tweak and text respectively (cf. Fig. 2b) to generate the keystream. When not using a TBC, the counter and PA are concatenated and then encrypted.

4.2 Memory integrity structures

A table of hashes or MACs protects against memory corruption, but it is not sufficient against replay attacks, unless the table is itself protected. This can be achieved by storing it in a tamper-averting memory or by covering it with a structure such as a *Merkle Tree* (MT) [Mer80] (cf. Fig. 3). MT nodes can be cached [GSC⁺03] to speed up verification.

With freshness-based encryption, we can protect the memory by just protecting the counters, for instance with a *Bonsai Merkle Tree*, i.e., a MT protecting the counter table [RCPS07]. A different method in the *counter tree*, of which examples are Hall and Jutla’s *Parallelizable Authentication Tree* (PAT) [HJ05] and the integrity tree used in SGX [Gue16a]. A node of the counter tree is called a *Counter Group* (CG). A CG contains a counters, which correspond to the a children of the node. The counters in a leaf, resp. non-leaf CG are one-to-one with a CLs, resp. children CGs. A MAC is computed on every

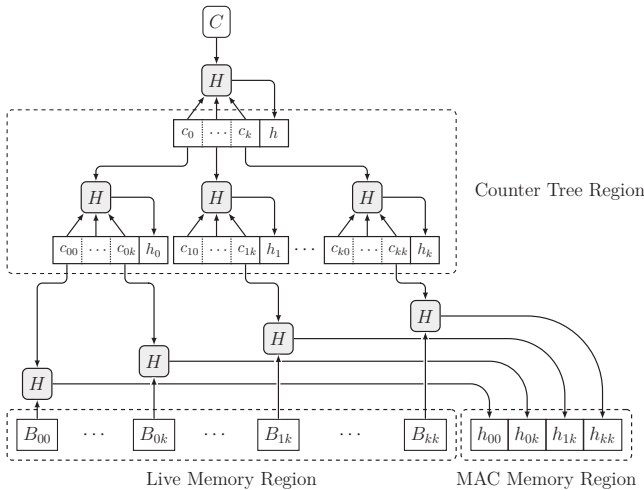


Figure 4: Counter tree for memory integrity.

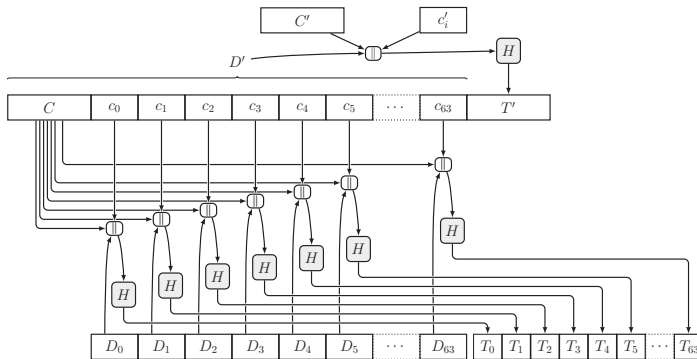


Figure 5: Split counters. The i^{th} logical counter is the concatenation of a major counter C (resp. C') and of the i^{th} minor counter c_i (resp. c'_i). This is then used in the function to compute the hash of the i^{th} data block (resp. counter group).

node and it is either stored dedicated table, along with the MACs of the data CLs or in the node’s CL along with the counters. Since the latter approach has better performance, for simplicity we consider only it. The MAC of a CG is computed on the a counters in the node and the parent counter. Before a node is evicted, its parent counter is first incremented and the node’s MAC is recomputed.

The *split counters* optimization [YEP+06] replaces a group of a counters with a group consisting of a single *major counter* and $a' > a$ smaller, *minor counters*, associated with that major counter (cf. Fig. 5). A *logical counter* in this scheme is defined as the concatenation of a minor counter and its associated major counter. Each node (a data CL or a CG) is associated with a logical counter. The increased arity (for instance, from $a = 8$ to $a' = 64$) reduces both counter storage overhead and tree depth. When a minor counter overflows, the common major counter is ticked to ensure that values do not repeat. Since this changes the values of all the logical counters associated with that major counter, all the sibling nodes need to be refreshed. For data CLs this means that they are re-encrypted, and for both types of nodes the MACs need to be recomputed. All minor counters in the group are reset to zero at this point to reduce the rate of minor counter overflows.

Delta encoding [YA18] works by representing a logical counter as the *sum* of the major counter and of a minor counter, instead of their concatenation. When a minor counter is about to overflow, the minimum of all minor counters m is computed, and if $m \neq 0$ then it

is subtracted from all minor counters and added to the major counter to delay the RMWs. This operation is also called *rebasing*.

Despite the RMWs, split counter trees bring a major performance improvement over monolithic counters. We introduce here 3-way split counters (with major, middle, and minor counters) to both increase arity *and* reduce RMWs.

Instead of using full trees, two optimizations can be done.

LoC One option is storing the data cache line counters in an in-package tamper-averting DRAM (a SRAM would be too large) which is MPE private (i.e., invisible to the rest of the system and outside adversarial control). We call this solution LoC which stands for *Leaves-on-Chip*. In fact, if we store the leaf nodes in a physically protected memory, such as on-chip, then we do not need to compute any other nodes from the original tree. LoC is sometimes mentioned in the literature only to be dismissed as unviable because of the large overhead.

BoC A less expensive version of the LoC solution consists of keeping the leaf nodes in external memory and store the level immediately above on chip. We call this tree arrangement BoC for *Branches-on-Chip*. Similarly to LoC, the system needs no further levels of the tree to ensure memory integrity. This idea seems new.

4.2.1 Memory overhead comparison

In Table 2, we compare memory overheads of different integrity trees, including the new very high arity trees introduced in this paper. Multi-CL MACs encrypt each CL individually with its own counter, whence the eviction of a CL from the last level cache does not require re-encryption of adjacent CLs. The table also includes the *Tamper-Evident Counter* (TEC) tree [ECL⁺07], which has high memory overhead and requires wide encryption with substantial latency, making it impractical for deployment.

4.2.2 Methods Excluded from Performance Evaluations

4.2.2.1 Log Hashes. *Log Hashes* [SCG⁺03b] are an interesting option because they do not employ a tree structure and have a negligible memory overhead. Log Hashes maintain an incremental hash of a Realm’s entire memory by adding the hashes of all cache lines in it. The hash of a cache line is computed on the concatenation of the contents of the line, its address, and a secret key. The Log Hash is updated with each memory write, by subtracting the contribution of the old contents, and adding that of new contents. Verification of the memory occurs only when the Realm interacts externally, but the Realm’s entire protected memory region needs to be verified. Log Hashes are well-suited only for long-running tasks with minimal I/O, where their performance impact can be negligible. They are unsuitable for general applications and remain undeployed in practice.

4.2.2.2 Morphable Counters. In the *Morphable Counters* paper [SNR⁺18a] various optimizations are presented to reduce the amount of RMWs associated to minor counter overflow. The first optimization builds on the observation that, for most tasks, either only a few counters in a group are incremented regularly, or all counters are. A counter group can have two representations, both intended as delta encoding. For a 64B CL length, all CGs contain a 64-bit embedded MAC, and a 57-bit major counter, a 1-bit format flag F and a 6-bit *Non-Zero Counters* (NZC) field. Then:

1. The first representation has $F = \text{“Zero Counter Compression”}$ and stores the number of non-zero counters in NZC. A 128-bit bitmap is used to mark which counters are non-zero, and the remaining 256 bits are partitioned in equal fields of size $\lfloor 256/\text{NZC} \rfloor$ each to store these counters.

Table 2: Memory Overhead of Various Types of Integrity Trees.

Legend: ℓ_H , ℓ_c , and ℓ'_c are the bit lengths of a hash or MAC; of a monolithic or major counter; and a minor counter, respectively. a is a counter group’s arity, and n is the number of CLs a MAC covers.

Type of Tree	CL Length	
	64 B	128 B
Merkle Tree with $a = 4$, resp. 8	33.3%	16.7%
<i>Monolithic Counter Tree with embedded MAC, $\ell_c = 56$</i>		
• $\ell_H = 64$; $n = 1$; $a = 8$, resp. 16	26.8%	12.9%
• $\ell_H = 32$; $n = 1$; $a = 8$, resp. 16	20.5%	9.79%
• $\ell_H = 32$; $n = 2$; $a = 8$, resp. 16	17.4%	8.23%
• $\ell_H = 32$; $n = 4$; $a = 8$, resp. 16	15.8%	7.45%
<i>Split Counter Tree with embedded MAC, $\ell_c = 64$</i>		
• $\ell_H = 64$; $n = 1$; $\ell'_c = 6$, resp. 7	14.1%	7.04%
• $\ell_H = 32$; $n = 1$; $\ell'_c = 6$, resp. 7	7.84%	3.91%
• $\ell_H = 32$; $n = 2$; $\ell'_c = 6$, resp. 7	4.71%	2.34%
• $\ell_H = 32$; $n = 4$; $\ell'_c = 6$, resp. 7	3.15%	1.57%
• $\ell_H = 32$; $n = 1$; $\ell'_c = 3$	7.04%	3.52%
• $\ell_H = 32$; $n = 2$; $\ell'_c = 3$	3.91%	1.95%
• $\ell_H = 32$; $n = 4$; $\ell'_c = 3$	2.35%	1.17%
PAT with $a = 8$, resp. $a = 16$	28.6%	13.3%
TEC tree with $a = 8$, resp. $a = 16$	42.9%	20.0%
128-ary 3-way Split Counter Tree, $\ell_H = 32$	—	3.91%
256-ary 3-way Split Counter Tree, $\ell_H = 32$	—	3.52%
128-ary 3-way Split Counter Tree with MirE, $\ell_H = 32$	—	0.78%
256-ary 3-way Split Counter Tree with MirE, $\ell_H = 32$	—	0.39%

2. If the number of non-zero counters is larger than 63, then the flag F is set to “Uniform” and the remaining 384 bits contain 128 3-bit minor counters. Rebasing is used only in this second representation.

While morphable counters are effective in reducing the number of RMW, we do not implement them since 3-way counters effectively eliminate the RMW overhead, and are much easier to implement (cf. Sections 5.8 and 5.11).

4.2.2.3 Isolated Tree with Embedded Shared Parity. We do not evaluate the *Isolated Tree with Embedded Shared Parity* (ITESP) [TBC⁺20] separately. One of its configurations packs 32 counters in a 64B cache line where the size of minor counters is 4 bits, and the freed 128 bits are used to store two 64-bit parity/integrity fields, each covering 16 cache lines. We speculate that its performance for a single Realm should be just slightly worse than a 64-ary 64B split counter groups L3 scheme with MirE, since no MAC table is kept. The main benefits of ITESP emerge when multiple Realms run concurrently, a configuration not supported by our setup, because each Realm would have its own integrity tree and metadata cache.

4.2.2.4 Approximate Set Membership. Attempts have been made to use Bloom Filters [Blo70] or Cuckoo Filters [FAKM14] for memory integrity schemes. The intention is to store the MACs of a group or CLs as *keys* in such probabilistic filters instead of storing them separately in memory. In order to achieve a *false positive rate* of at most p , Bloom filters, resp. Cuckoo filters, need at least $1.44 \log_2(1/p)$ [PPR05], resp. $\log_2(1/p)$ bits of space per inserted key [FAKM14]. Since we want the false positive rate to approach, if not match, the claimed security level of the MACs inserted as keys in the filters, which is their bit length,

these filters cannot occupy less memory than an equivalent table of MACs.¹ Therefore, all the proposed schemes that claim a memory overhead reduction [NHSQ12, SA21] fail this basic security scrutiny.

4.3 Cryptographic parameters and practices

To ensure long-term confidentiality, *encryption keys should be at least 128 bits long*. Shorter keys are not used in any currently deployed or recently proposed memory protection scheme. Sometimes longer keys are an option, for instance 256-bit keys for Intel’s TDX, but we posit that this does not offer increased practical security and only increases latency: indeed, a proper complexity analysis of quantum-computer-assisted key search against AES-128 proves it is secure even against adversaries with access to a large-scale quantum computer [JNRV20]. Deployed technologies such as Intel’s SGX and TDX, and AMD’s SEV use the AES in modes that need two independent keys, or even AES-256. QARMA-128 and QARMAv2-128 allow the use of 256-bit keys as well.

Encryption block sizes must be at least 128 bits, to reduce the likelihood of any attack that exploits ciphertext collisions.

Authentication keys should be at least 128 bits long as well.

Only the TCB and no SW environment may set any key, and SW will only manage process identities.

We posit that a length of 32 (or even 28) bits is sufficient for both data and counter group MACs, to deter Adversaries that simply want to corrupt memory, for instance with RowHammer attacks. This is, in fact, one of the main reasons to deploy a L2 scheme. The TCB must destroy (i.e., internally invalidate and overwrite) any differentiator, i.e., key or tweak, associated with the address where an integrity violation occurred — and possibly other internal information. The target process will no longer be able to execute, and the information in it will be lost to the adversaries. It is essential that the TCB responds so to integrity violations before giving back control to the operating system or the hypervisor. Otherwise, to make just one example, an $\mathcal{A}_{\text{active}}^{\text{HW}}$ adversary with the ability to run privileged SW would be able to brute force a short MACs.

If the chosen authentication primitive produces a longer MAC than needed, the output is simply truncated.

In L3 schemes, an Adversary may attempt to replace a CL together with its MAC. To do this without triggering an integrity fault, they wait until the counter associated with the target CL repeats. If the counters are sufficiently long, the attack cannot succeed. For this reason, monolithic counters must be at least 64 bits long (it can be argued that 56 bits suffice). The minimal aggregated length of a major and a minor counter (or major plus middle plus minor) shall also be 64 bits. If an Adversary wants to replay a CL together with its MAC and counter, they will similarly have to either guess the embedded MAC or wait that the parent counter repeats.

For Merkle Trees the minimal hash length is 128 bits, regardless of whether the hash is keyed or not, to ensure that attacks have a time complexity of at least 2^{64} .

4.4 On the design space

In Fig. 1 an MPE is associated with a memory channel, benefitting from memory interleaving and thus reducing bandwidth saturation risks. In the figure an MPE is also represented as a separate block between system cache and memory controller, but this is far from the only option: it can be implemented as part of the memory controller or a wrapper around the system cache. A different MPE configuration involves a core-private MPE, positioned

¹Entropy is a harsh mistress. Attempts to cheat on her are severely punished by mathematics.

upstream of the on-chip interconnect. In such a design, the MPE can be a performance bottleneck, but it is suitable for *secure cores*, like SoC-embedded TPMs.

Pure SW solutions are possible: At boot, a part of a cache is *address locked* in order to keep the TCB in it (and effectively reducing its size). All memory reads/writes to external memory are then trapped to this code to augment them with encryption and integrity support. Performance is clearly severely impacted, as in [CZG⁺15, MMS⁺20]. A different, less secure, approach [Pet10, GMD⁺16] keeps most of the RAM encrypted except for a few recently used pages, which are re-encrypted once they have been idle for some time.

Recall that we only consider solutions contained in the SoC package. This excludes any form of “smart memory” [AN17] where the protection logic is split between the Requester and the Completer, such as the CXL *memory Integrity and Data Encryption* (IDE) scheme [CXL19]. Such architectures require logic for attestation, secure link setup, and encryption, involving cryptographic engines in every memory module if not every chip, so it would be more expensive, hardware-wise, than an MPE-based solution. CXL is however suitable for disaggregated memory configurations, covering transport between compute and memory nodes.

The breadth of the subject and constant developments (cf. Table 1) imply that the full design space is likely not knowable. The present work represents just a snapshot.

5 Benchmarking plan, results, and discussion

5.1 Benchmarking environment and methodology

It would be impractical to implement several thousands of combinations of technologies in silicon for the purpose of evaluating them. A solution to this problem lies in prototyping, i.e., the creation of an approximate implementation of the desired features, which can thus be tested and benchmarked. Very accurate models can be created even without implementing all details. For instance, the latencies of cryptographic primitives can be derived from actual implementations and inserted as delays into the simulation.

The prototypes used in this paper are built in the `gem5` simulator [BBB⁺11, LAA⁺20]. `gem5` allows engineers to build SW versions of HW components typically included in computer systems. It abstracts the interfaces between components, which can be combined flexibly. It provides approximate timing models for many processor cores.

The modeled CPU core approximated an Arm Cortex A72, with a 2 GHz frequency and a 1 GHz system frequency. The cache hierarchy includes L1-I (48 KiB, LRU replacement policy, 3-way set associative, 1 cycle latency) and L1-D (32 KiB, LRU replacement policy, 2-way, 1 cycle latency) caches, and a unified L2 cache (1MiB, tree-PLRU replacement policy, 16-way, 5 cycles latency). The memory is 16 GiB DRAM in a dual-rank DDR4 DIMMs configuration. The MPE-private caches are 4-way set associative with a LRU replacement policy.

The simulated SoC is implemented in a 7 nm process. We take the latencies of some components from [Ava17], for instance 15.67 ns for a pipelined implementation of AES-128, 4.8 ns for QARMA₁₁-128- σ_1 and 2.2 ns for QARMA₅-64- σ_0 . Note that implementation, process, libraries all affect the crypto block’s latency, but system and CPU clocks do not. We assume we reuse the IP blocks from [Ava17] with their own clocks, thus with the exact same performance characteristics. This is a reasonable assumption since this is how hard macros are used in practice. The above latency of QARMA₅-64- σ_0 is also used in [JLK⁺23], and essentially for the same purpose as ours.

Lastly, all MPE algorithms are thoroughly parallelized to their maximum extent for all considered schemes.

Our evaluation uses the SPEC 2017 [BLvK18] benchmark suite. Detailed software models such as `gem5` increase execution time by several orders of magnitude: a typical

SPEC benchmark can take around a month to run [San14]. To facilitate rapid prototyping, we use the SimPoint [SPHC02] methodology, which is well understood in academia and industry. It uses clustering to find representative regions that serve as a proxy for the whole application. The results are finally combined using weighted averages, that reflect the regions' importance to the overall application. Up to 10 SimPoints of 30 million instructions from each benchmark are simulated in place of several billions of instructions. Regarding reproducibility, including all details needed to re-generate our SimPoints would be impractical. We also note that the SPEC consortium themselves encourages the use of SimPoints [SPE18].

Remark 4. An alternative approach would have been to run the entire benchmarks, as opposed to SimPoints, in parallel on a large distributed cloud. This unfortunately does not work in practice since the longest running workloads would have taken weeks to months to run to completion, without any expectation that the results would be any different from the SimPoints methodology. The quicker turnaround, less than an hour to run all SPEC 2017 on a big-enough cluster, is in fact instrumental when exploring a vast space of optimizations.

Remark 5. A legitimate question is whether we can verify the reliability of our simulations by porting SPEC2017 to run under Client SGX. This would be a major undertaking, even with the help of general-purpose wrappers, we would have to avoid the penalties related to the EPC. In fact [Gue16a] only runs 445.gobmk from SPEC2006 with selected data sets. This said, on the `trevorc.tst` and `nngs.tst` data sets, [Gue16a] reports slowdowns of 4.90% and 3.29%, respectively, and on our simulated SGX-like method we measure 5.31% and 4.65%, in line with expectations from a deeper tree.

Remark 6. Regardless of how the simulation is performed, we may ask ourselves about the impact on systems that include context switches, virtual memory swap, and any type of I/O. These aspects are very difficult to emulate. In fact, benchmarking in such a context seems absent from the literature on cryptographic memory protection. However, (i) We can observe that the additional memory used for metadata is not visible to the operating system and will be unaffected by paging and similar operations; and (ii) It can be argued that context switches, paging, and general I/O are affected by the performance penalties on memory accesses only in a minor way: context switch code and data can reside in pinned memory, and the timing of disk, network operations is dominated by media which are orders of magnitude slower than physical RAM. Speaking in particular of context switches, on a typical many-core system (for instance 128-core) with several users (roughly 50 users and up to 500 active user sessions), we observe less than a handful of context switches per second per core. Any cold start effect after the context switch would be in the noise since warming all the caches take just a few million instructions (roughly a few milliseconds).

Therefore, any performance penalty we present here is likely an upper bound to the real-world one.

5.2 Selection of the benchmarking sets

All MPE configurations span a vast multidimensional space. Exhaustively evaluating them all is clearly infeasible, not to speak of the difficulties of properly presenting the data. Hence, we explore the design space in various stages, each consisting of a *set* of runs of the benchmark suite. Each set focuses on some previous configurations and expands the parameter space *where we expect that it has some noticeable impact*. Some schemes, such as L1 schemes, do not carry over to the successive sets because they do not have implementation parameters beyond the encryption primitive.

We use shorthands to describe the various configurations:

Level / Cipher / {additional technologies} / MAC length / CL length .

The optional “additional technologies” may include: counter representation (mono or split) and arity, Leaves or Branches on Chip (LoC or BoC), or the use of MACs in Repurposed ECC bits (MirE).

The default CL length is 64 B, unless the counter groups are on chip, in which case it is 128 B. The default MAC length is 56 to 64 bits.

“{AMD} SME” is equivalent to L1/AES/GFmul/CL64B. Here, GFmul denotes a XEX scheme where the tweaking mask is computed by multiplication of the tweak by an additional secret key, whereas when we just write XEX the mask is derived by encryption of the tweak; “{Intel} TDX” is equivalent to L2/AES/MirE/28b/CL64B, and “{Intel} SGX” is based on Client SGX, i.e., L3/AES/mono-8/56b/CL64B. LoC always implies counters are split. L2 implies that a non tweakable block cipher is used in a XEX construction, except when explicitly stated otherwise. The shorthand L3/LoC denotes a version of L3 that uses LoC, and thus no integrity tree. Similarly, L3/BoC is a L3 solution with the leaf counters off chip and the next level on chip, also without a full tree. L3 *without* BoC or LoC denotes a replay-protection-capable scheme based on an integrity tree and *no counters on-chip*.

5.2.1 Simulation of system load

The benchmarks are first run on an *unloaded* system, where the current benchmark is the only running task.

We then want an upper bound for the performance degradation in a fully *loaded* system, with up to hundreds of processes running on dozens of processing elements, all sharing the bandwidth of the memory subsystem, such as in a cloud server. Directly simulating such a system is very complex and impractical. We instead inject synthetic traffic upstream of the MPE, but after the L2 cache. We do not include a L3 cache in the system to simulate the extreme situation where the latter has been completely swamped by traffic coming from other requesters or clusters of requesters.

The question is then, how much extra traffic we must inject.

Therefore, we measure the effective memory latency of the system with various levels and schemes of memory protection, and we observe that the latency starts to degenerate catastrophically for most of them between 8 and 10 GiB/s. Fig. 6 shows how latency, and thus, at least part, also performance penalty depend on the load of the system. For instance, a SGX-like L3 MPE covering the entire memory starts to degrade if more than 8 GiB/s of traffic is injected. We take this value as the traffic for a fully-loaded system and halve it, i.e., 4 GiB/s for the partially-loaded system.

The simulated traffic consists of 75% reads and 25% writes of entire cache lines (64 B or 128 B). The access pattern is a mix of cache-line-aligned linear and random accesses. The linear accesses are sequential, and the random ones are at randomly generated addresses, both across the whole reserved range. The traffic generator alternates 100 μ s of simulated time of linear accesses with 200 μ s of random accesses, for as long as the workload is running.

5.2.2 Baseline performance

Without memory protection, our benchmarks run on a loaded system 14.1% slower than on an unloaded system with 64 B CLs, and 9.5% slower with resp. 128 B CLs. Changing the CL length from 64 B to 128 B results in an average speedup of 1.4% in an unloaded system and 5.5% in a loaded system. The timings of all benchmark runs are always compared to the baseline with the same load and CL size.

5.2.3 Initialization of short counters

When a piece of software starts to run, in a real-world setting any minor/middle counter will have assumed, because of previous processes, essentially random values. If all counters

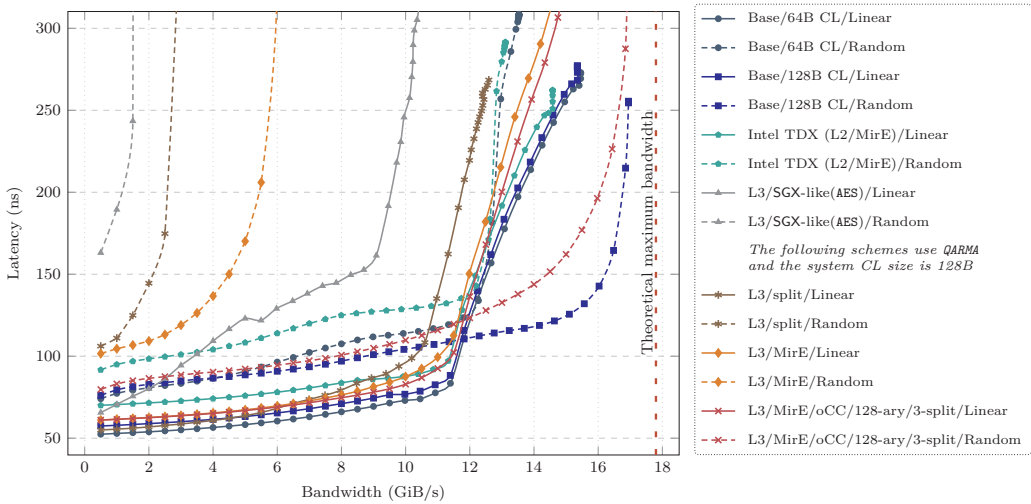


Figure 6: Bandwidth/latency plot with various MPEs and without, for linear or random synthetic traffic.

are initialized to zero before running a benchmark, the latter is put at an advantage, since the non-major counters will take longer to overflow, and the number of RMWs may be underestimated. In fact, SimPoints may even amplify this bias. Therefore, to make our simulations as realistic as possible, in all split counter runs we initialize the non-major counters to uniformly random values. This magnifies the performance gap between 2-way and 3-way split counters of equal arity, highlighting the superiority of the latter.

We now report and discuss the results of all the runs.

5.3 Set 1: State-of-the-Art, AES vs. Lightweight encryption ciphers, and 64-bit vs. 32-bit MACs

We start with the state-of-the-art and some simple variations thereof to get an initial overview of the relative performance merits of the deployed or proposed technologies. We compare L1/AES/CL64B (e.g., AMD SME), L1/QARMA/CL64B, L2/AES/32b/CL64B, L2/AES/MirE/28b/CL64B (corresponding to the Intel TDX and Scalable SGX MKTMEi), L2/QARMA/32b/CL64B, L2/QARMA/MirE/32b/CL64B, and ELM with both monolithic and split counters, SGX (i.e., memory protection as in Client SGX, but covering all memory, L3/QARMA/split-64/32b/CL64B — all with and without a hash cache if not fixed by the manufacturer’s architecture, since some architectures have a hash/MAC cache while other ones, such as SGX, avoid it. We also compare 32-bit and 64-bit MACs in selected cases — shortened to 28-bit, resp. 56-bit, in TDX, resp. SGX.

Note that SGX here is not a full implementation of Intel’s Client SGX architecture, but only of its encryption, integrity, and anti-replay features, the latter expanded to the whole memory. For SGX, hash encryption is CTR as described by Intel [Gue16a]. We use this method for the SGX-like variant with AES-256 (L3/AES256/mono-8/56b/CL64B) as well. In all other cases, data MACs are replaced by 32-bit long hashes which are directly encrypted in groups of four upon eviction.

Note that TDX includes also Scalable SGX.

The ELM method follows [IMO⁺22], i.e. it uses the AES in a XEX construction except when QARMA is used. With QARMA the XEX constructions are replaced by simply feeding nonces and separation fields as the tweak to QARMA, as well as using QARMA_{5-64-σ₀} to generate the *One-Time Pads* (OTPs) to encrypt the tags. We note that Apple’s Secure Enclave uses a method that is essentially equivalent to ELM’s for encryption, but integrity

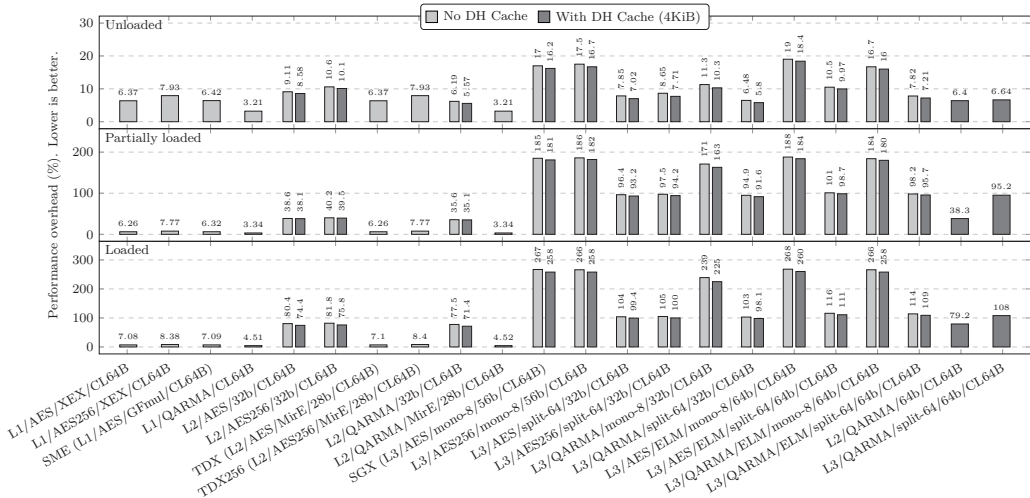


Figure 7: Set 1 (Section 5.3). Comparison of base levels and state-of-the-art.

is computed with CMAC, which is more expensive than encrypted checksums and does not lead to enhanced security. Since it would probably be just slightly slower than ELM, we do not need to benchmark it.

Monolithic counter trees are 8-ary, resp. 16-ary with 64 B, resp. 128 B CLs. For 2-way split counters, minor counters are always 6, resp. 7 bits long, and the arity is 64, resp. 128.

For schemes that provide freshness, the counter cache is 64 KiB as in SGX to level the comparisons.

These principles apply to every successive set as well, unless otherwise indicated.

The runs reported in Fig. 7 support Results **R2** and **R5**. Also, ELM has worse performance than SGX, having the encryption primitive on the critical path.

Recall that the latencies of AES-256, AES-128, and QARMA-128 in our simulation are 21.99 ns, 15.67 ns and 4.80 ns, respectively. They are strongly correlated to the corresponding performance penalties of a L1 scheme on an unloaded system: 7.93%, 6.37%, and 3.21%. Similar outcomes hold also for varying loads and L2/MirE schemes. For L2/non-MirE and L3 schemes, the difference becomes less significant as the slowdown due to traffic contention between data and metadata increases. This proves Results **R3**.

We observe that the two XEX schemes L1/AES/GFmul/CL64B and L1/AES/XEX/CL64B have the same overall performance penalty. Indeed, Fig. 13 shows that this is the case for each individual benchmark. These schemes differ only in the computation of the tweaking mask. In the first case it is performed via Galois multiplications, which we highly optimize for speed, resulting in a latency of 0.55 ns in the chosen process. In the second case AES encryption is used instead. We recall that AES-128 latency is 15.67 ns. Thus, on the write path, the latency is, roughly one, resp. two AES instances, while on the read path is it always one AES instance. Despite the significant difference on the write path, the penalties are almost exactly the same. This supports the first claim in Result **R4**.²

To a first approximation, the relation between performance penalty and additional read latency should be roughly a linear function of the relative additional latency $\epsilon = \alpha\chi/\beta$ where β is the baseline memory latency, χ the additional read latency, and α depends on the proportion of external memory accesses while the task at hand is being run. While we can estimate β (cf. 6), α is clearly too variable. Next, we apply least squares regression to the latencies of AES-256, AES-128, and QARMA-128 and the corresponding performance

²In passing, we mention that this aligns with common lore that the read latency is the dominant factor on the overall performance impact.

penalties for a L1 scheme on an unloaded system. The penalties roughly lie on the line $0.2765\beta + 1.922$ with errors bounded by 0.11. The middle value is underestimated, suggesting a positive coefficient for the second order of the approximation. This completes our arguments for Result **R4**.

*For the remainder of the evaluation, because of Result **R5**, for simplicity's sake we shall assume that MACs are 32 bits long and directly encrypted in groups of four except with SGX, MirE, or otherwise explicitly indicated.*

*Similarly, since split counters perform better than monolithic counters (this goes towards Result **R11**), we shall assume that L3 configurations will make use of split counters.*

For brevity, in Sets 2, 3 and 4 we leave out AES from the comparison as it has an identical memory access pattern and similar results to using QARMA-128.

5.4 Set 2: Impact of MPE cache sizes

The goal here is to understand the impact of the sizes of the two MPE caches, namely the hash and counter caches.

L1 does not need caches, so we only consider L2 and L3.

The hash cache sizes we evaluate are 4 KiB, 16 KiB, and 64 KiB; and counter cache sizes are 16 KiB, 64 KiB, 256 KiB, and 1 MiB. We expect these sizes to be within a reasonable range when implemented as SRAM. The presented results are based on the two L2/QARMA/32b/CL64B and L3/QARMA/split-64/32b/CL64B configurations (i.e., 32-bit MACs, 64 B cache lines, and 64-ary split counters for L3).

These results, displayed in Fig. 8 support Result **R6**. The small benefit of the hash cache can mostly be attributed to spatial locality (most temporal locality has already been exploited by normal data caches). Intuitively, the access patterns of the counter and the hash cache should be similar. However, the reach of the counter cache is bigger since counters are smaller when using split counters and nodes closer to the root cover a large amount of address space which makes them more likely to be reused.

Starting with Set 3, the MPE has a 16KiB hash cache and a 256KiB counter cache. Level L3 uses split counters, unless explicitly indicated otherwise, or with SGX.

5.5 Set 3: Impact of the cache line length

Another fundamental piece of information is how the choices of 64 B and 128 B CLs affects L2 and L3 performance: Doubling the CL size will halve the memory overheads, but at least in theory the coarser memory granularity may negatively affect performance.

This set comprises L2/QARMA/32b and L3/QARMA/split/32b with 64 B and 128 B cache lines. Counter group and CL sizes are always equal which implies that the basic L3 split counter configurations have arity 64 in the 64B case and 128 in the 128 B case.

The results of Set 3 are combined with those of Set 4 in Fig. 9. They prove Result **R7**. Since we already know that our reference system without an MPE performs 1.4% to 5.5% better with 128 B CLs, we expect that using to 128 B CLs, at least for the system cache, is generally beneficial in a system with an MPE.

It is worth noting that having shorter CLs in coherent caches closer to the CPU is still possible and may be beneficial for multithreaded workloads but studying this is outside the scope of this paper.

Remark 7. Changing the cache line size for the coherent cache system can be a major undertaking. However, there are important cases where it is not only feasible, but also reasonably non-intrusive. For example, inclusive *Last Level Caches* (LLCs) could store and perform writebacks of pairs of 64 B cache lines while still performing coherence on the individual lines. Similarly, LLCs outside the coherent domain (sometime known as *system*

caches) may use 128 B cache lines while the coherent caches use 64 B cache lines. Both options make the effective cache line size 128 B from the point of the MPE.

5.6 Set 4: Asynchronous MAC verification

So far, we have assumed that integrity tags are verified synchronously. In principle, asynchronous verification can improve performance by releasing data to the CPU before its MACs has been fetched from memory and verified. Therefore, we assess how synchronous verification improves overall performance over asynchronous verification.

We test only L2 and L3, as they offer integrity. We reuse the configurations of Set 3. The results are shown in Fig. 9.

Asynchronous verification comes with a significant drawback. Since the CPU is speculating on MAC verification being successful, adversaries have a window of opportunity where the CPU is using data under their control and mount an attack. Mitigating this issue introduces significant complexity which would be detrimental the integrity of the system. This is Result **R8**.

From here, we only use synchronous MAC verification.

5.7 Set 5: Use of on-chip memory for L2 and L3

Going beyond caching as explored in Set 2, we explore the impact of secure MPE-private on-chip memory.

Since MACs have a larger memory overhead than counters, we do not expect schemes with on-chip hashes and off-chip counters. Hence, we ignore such a configuration.

Fig. 11 results confirm that relieving the memory bus contention between data and metadata improves performance already when the memory subsystem is partially loaded.

The BoC configuration only marginally outperforms the schemes that do not rely on on-chip memory. This is explained by considering a system without on-chip memory: Temporal locality is poor for leaf nodes, but it improves closer to the root of the tree as each node corresponds to a large memory space. This makes it likely that integrity verification encounters a cache hit at the level just below the leaf level. Therefore, performance is similar to BoC.

With all metadata on chip, the performance is close to the baseline. This may not be realizable in practice. However, as we shall see in Section 5.8, it can be approximated by repurposing ECC bits for MAC storage.

For this set of runs we kept the AES to show that for L3 the performance is similar to QARMA. However, on an unloaded system, AES and QARMA show a slight performance gap. This gap decreases as the system load increases, due to the fact the cipher latency becomes proportionally smaller compared to the increasing memory access latency.

5.8 Set 6: Impact of repurposing ECC Bits, 3-way split counters, and large counter caches

The deployment of Intel TDX’s Multi-Key Total Memory Engine with Integrity (MKTMEi) [Int21] and [YA18] suggests that using ECC bits for tags may be an acceptable trade-off for real-world deployments. This is essentially an approximation of storing MACs on-chip since the ECC bits are stored out-of-band and fetched in parallel with the data.

We consider both L2 and L3 configurations, with and without MirE. We expect that MirE implementations are optimized for performance and to reduce storage overhead. For that reason, except for L1/QARMA/MirE/CL64B, we focus on 128 B CLs which enable denser counter packing than 64 B CLs. With MirE the MAC algorithm is PMAC, and a hash cache is not needed since MACs and data are fetched in the same memory transaction.

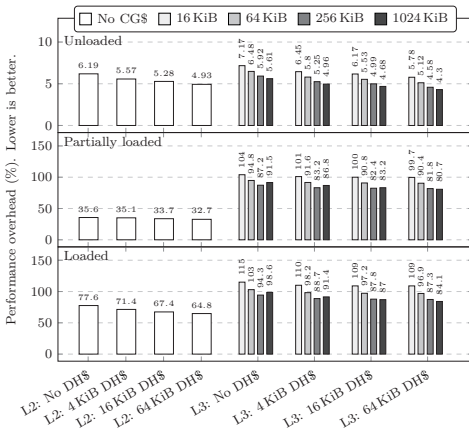


Figure 8: Set 2 (Section 5.4). Impact of MPE cache sizes on L2/QARMA/32b/CL64B and L3/QARMA/split-64/32b/CL64B.

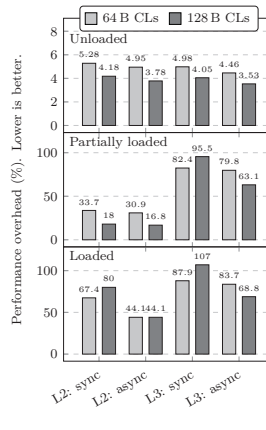


Figure 9: Sets 3 and 4 (Sections 5.5 and 5.6). Impact of CL size and asynchronous MAC verification.

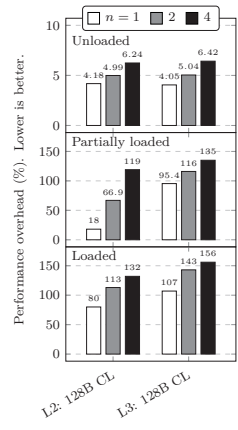


Figure 10: Set 7 (Section 5.9). Impact of incremental MACs.

In addition to classic 2-way Split Counter Groups (CGs), we introduce high-arity 3-way Split CGs, which we define only in the length of 128 B, with 128 and 256 logical counters per node. The purpose of this optimization is to keep the amount of RMW operations under control, with one variant also increasing the density of the CGs. To quantify the impact of this optimization, we evaluate each configuration with and without middle counters. We consider the following 3-way split CG types, without embedded MACs:

- 128×7 -bit minor, 8×8 -bit middle, and 1×64 -bit major counters, with a memory overhead of 1:128; and
- 256×3 -bit minor, 32×6 -bit middle, and 1×64 -bit major counters, with a memory overhead of 1:256.

If MACs are embedded in the counter group, for a 128-ary tree the lengths of the major and middle counters would be reduced to 48 and 6 bits, and for a 256-ary tree the middle counters would 5 bits long — in both cases with 32-bit MACs. The memory overheads of these trees are 1:127 and 1:255, respectively.

The data (Fig. 12) supports Results **R9** and **R11**. Middle counters play a crucial role in maximizing the performance of high arity CGs, preventing significant RMWs overheads. This demonstrates Result **R12**. Because of this, L3/MirE/LoC designs may even perform better than L1 schemes, which have the cipher on the critical path to the external RAM.

For a 16 GiB protected memory, the BoC configuration needs 256 KiB of on-chip storage. An alternative to the BoC configuration would be to use that memory for a counter cache. The 512 KiB configuration in Fig. 12 corresponds to this configuration since the baseline counter cache size is 256 KiB. In such cases, the larger cache normally performs on-par with BoC in an unloaded system and slightly better under load. This can be explained by two effects. First, the cache approximates BoC since the level just above the leaf level is very likely to be resident in the cache. Second, unused branch nodes can be replaced by useful leaf nodes which improves efficiency. On a fully loaded system, LoC performance is reached in practice only when the cache is large enough to cover the tree working set of the running applications. In the case of SPEC 2017, this typically happens between 1 MiB and 2 MiB of cache.

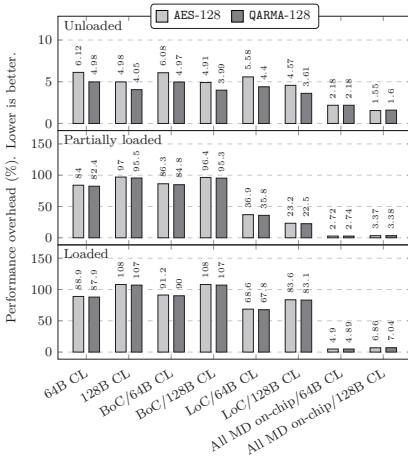


Figure 11: Set 5 (Section 5.7). L3: Impact of storing metadata on-chip.

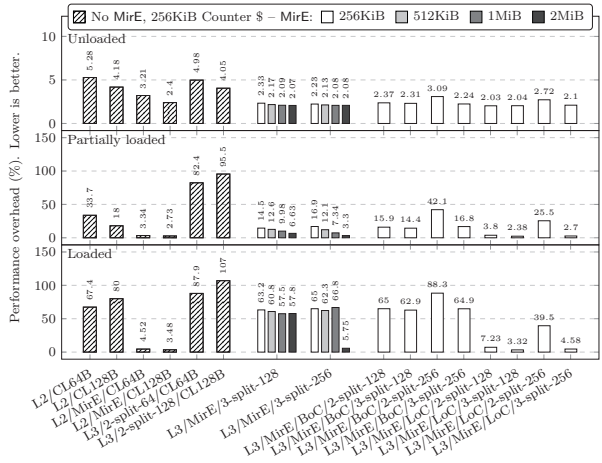


Figure 12: Set 6 (Section 5.8). {L2|L3}/QARMA/MirE: Impact of repurposing ECC bits for MACs and large counter caches. For arity 128 or 256 the CL is always 128 B.

5.9 Set 7: Impact of incremental MACs

If we cannot store MACs in the ECC bits or on-chip, there is another option for reducing their storage overhead: to compute them incrementally over multiple cache lines.

Since the goal here is to reduce storage overhead, we consider only 128 B CLs. We test both L2 and L3 configurations with a MAC covering 1, 2, or 4 CLs. The runs are reported in Fig. 10. We use only QARMA-128 for encryption, since the performance degradation depends only on the increased memory traffic. In fact, AES results follow the same pattern.

To improve the performance of an incremental hashing scheme one can use the following optimization, presented in [TSB18]: Whenever the plaintext is sufficiently compressible, store the ciphertext along with its MAC in the same CL, and do not include the MAC in the computation of the incremental MAC of the surrounding CL group. This reduces the number of memory reads and the amount of operations to update the incremental MAC. However, this optimization comes with a significant drawback: an Adversary can infer properties of the data by observing its compressibility, defeating the purpose of confidentiality protection [Kel02, SBS⁺21]. Therefore, we do not consider it.

This is Result **R10**.

5.10 Set 8: Breakdown of selected configurations

To better understand the behavior of the MPE, we select a few interesting configurations and show all individual benchmarks in the suite:

- AMD SEV (L1/AES/GFmul/CL64B) and L1/QARMA/CL64B;
- Intel TDX (L2/AES/MirE/28b/CL64B);
- L2 with (L2/QARMA/MirE/28b/CL64B) and without (L2/QARMA/64b/CL64B) MirE;
- Intel SGX (L3/AES/mono-8/56b/CL64B);
- 128- and 256-ary 3-way split CGs (L3/QARMA/LoC/3-split-128/MirE/28b/CL128 and L3/QARMA/LoC/3-split-256/MirE/28b/CL128).

The SPEC2017 benchmarks (cf. Figs. 13 to 18) exhibit some expected results: certain tasks, like `omnetpp`, `mcf`, and `bwaves` experience a more significant performance impact across most MPE configurations.

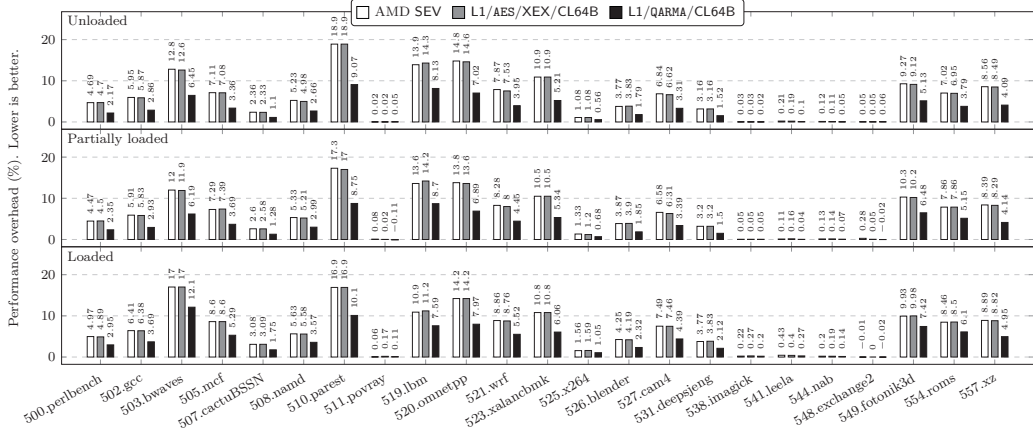


Figure 13: Set 8 (Section 5.10). Comparison of AMD SEV (L1/AES/GFmul/CL64B), L1/AES/XEX/CL64B, and L1/QARMA/CL64B.

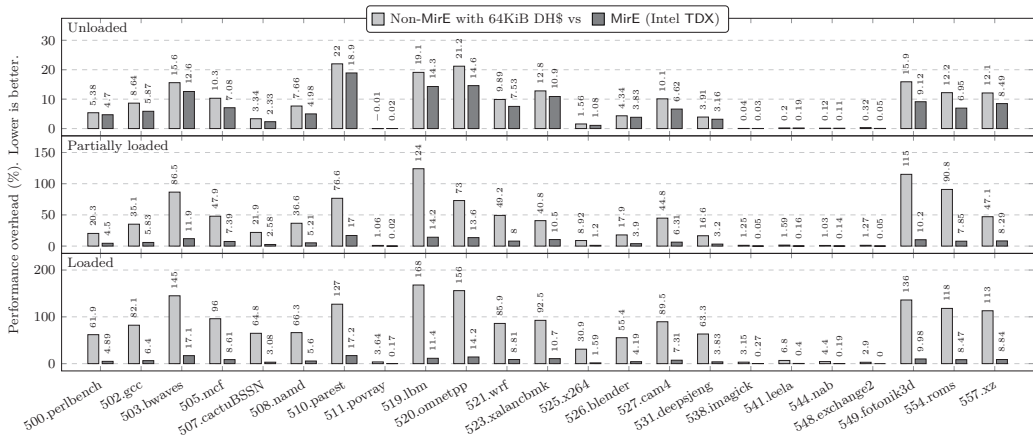


Figure 14: Set 8 (Section 5.10). L2 impact of MirE: L2/AES/32b/CL64B vs. L2/AES/MirE/28b/CL64B (e.g., Intel TDX).

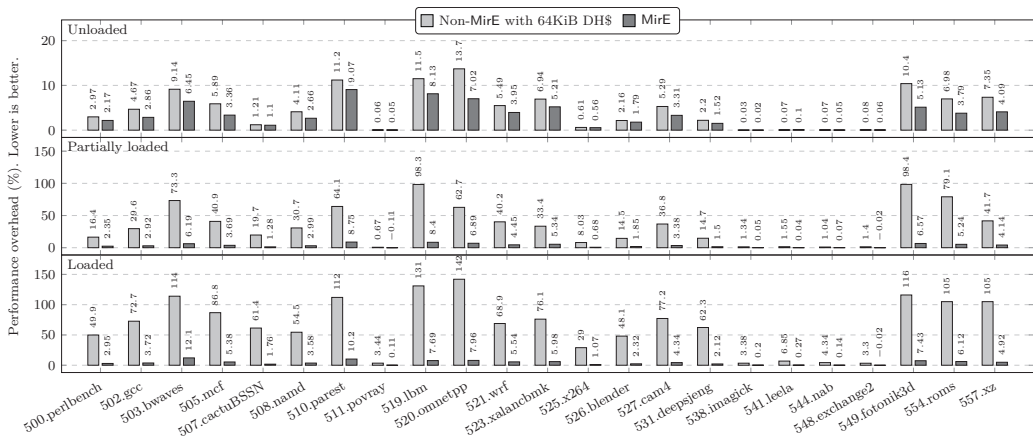


Figure 15: Set 8 (Section 5.10). L2 impact of MirE when using QARMA: L2/QARMA/32b/CL64B vs. L2/QARMA/MirE/28b/CL64B.

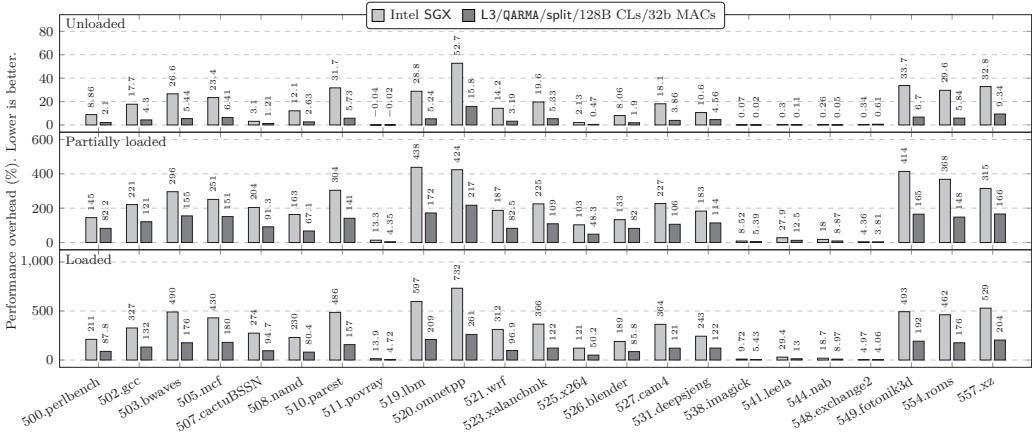


Figure 16: Set 8 (Section 5.10). Impact of split counters: L3/AES/mono-8/56b/CL64B (Intel SGX) vs. L3/QARMA/split-128/32b/CL128B.

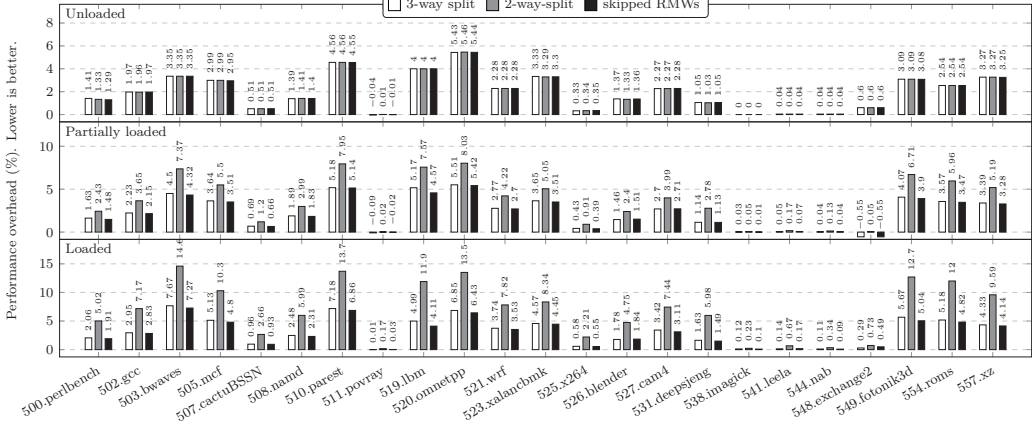


Figure 17: Sets 8 and 9 (Sections 5.10 and 5.11). L3/QARMA/MirE/split-128/LoC/28b/CL128B with 3-way and 2-way split counters, and without RMWs.

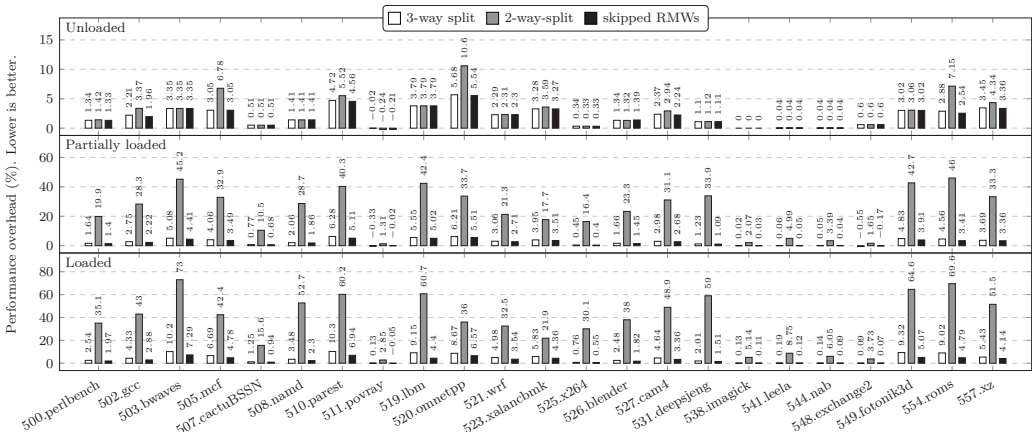


Figure 18: Sets 8 and 9 (Sections 5.10 and 5.11). L3/QARMA/MirE/split-256/LoC/28b/CL128B with 3-way and 2-way split counters, and without RMWs.

5.11 Set 9: Impact of RMW operations

All split counter methods need, as already mentioned, to perform some batches of RMW operations to re-encrypt data or re-compute some embedded MACs whenever a minor, resp. middle counter overflows. These are expensive operations and we want to understand their impact on performance.

We compare the performance of L3 MPEs against hypothetical ones where the RMW operations have zero cost, i.e., are instantaneous. This is achieved by simply skipping them: such an experiment is possible because the simulated MPE does not actually perform cryptographic operations, inserting instead timing delays in their places. This gives an upper bound on the actual time spent in the RMW operations.

For the 128- and 256-ary split counter schemes, we report the performance with 3-way split counters, the performance with 2-way split counters by omitting the middle counters, and the performance with skipped RMWs. The selected combinations are the ones in Set 8 with RMWs.

The results are shown in Figs. 17 and 18. We notice that the impact of RMWs is not always negligible. Using 2-way split counters with 3b minors (L3/QARMA/LoC/MirE with 256-ary CGs) carries a significant performance penalty, but the use of middle counters brings the performance close to the ideal case where RMWs are “free”.

The performance penalties and the proportion of time spent doing RMWs increase with the load.

This set of runs proves Results **R1**, **R11**, and **R12**.

5.12 Remarks on area and power

Power consumption of a circuit is roughly a linear function of both its area and the time it is active.³ Thus, the MPE’s total area and the performance penalty are the main factors determining its energy cost.

The area of the MPE mostly consists of arithmetic circuits, caches, and any internal DRAM (if present). In comparison, the control circuitry has negligible area.

Not only is estimating areas for all configurations impractical, but also implementations can vary greatly. For direct encryption schemes like L1 and L2, implementing multiple encryption blocks in parallel maximize performance, but area can be saved by sacrificing some of that performance using pipelined designs. An area-optimized implementation of QARMA-128 (with 256-bit keys) is roughly ≈ 50 KGE for a single pipelined block [Ava17]. Latency-optimized AES implementations exceed 17 KGE per round [UHM⁺20], hence the area for a single instance is ≈ 160 KGE and for eight parallel blocks ≈ 1.3 MGE. Note, also, that a pipelined QARMA circuit and a fully parallelized AES circuit would have comparable total latency — and this would deliver similar performance and security to a L1/L2 scheme, while having different areas.

Integrity can be implemented by re-using the encryption blocks, or by using ad-hoc, smaller ones like QARMA_{5-64- σ_0} , as in [JLK⁺23].

Remark 8. The speed of the MPE caches is not as critical as the CPU caches’: counters and hashes just need to be available to the MPE before the data from RAM. Thus, instead of SRAM, slower DRAM with a much smaller area can be used for these caches.

A DRAM memory cell uses a capacitor and transistor, or in some cases two transistors. The area can thus be capped by two transistors per bit, with a minor amount of control logic. A 4 KiB cache is thus about 64 KGE, and a 256 KiB cache is roughly 4 MGE.

With these numbers at hand, we see that, for modern SoCs with billions of transistors, a single large MPE per memory channel is a small but not negligible cost. Although an

³To be more precise, power consumption is the sum of dynamic power, that depends on switching current, and static power, that depends on leakage current, and thus on power gating.

additional 1:128 or 1:256 of in-package or in-module DRAM might seem a minor investment compared to the total system memory, it cannot be disregarded, especially considering the added expenses of tamper-averting designs.

6 Conclusions and Open Questions

We performed a thorough survey and evaluation of the available technologies for the cryptographic protection of memory contents, together with some previously not considered variants. This results in a vast set of independent choices, corresponding to different prices in terms of have each their performance penalty, memory overhead, and hardware cost. The lack of an absolute metric to combine these three costs into one rating makes it very challenging to provide recommendations for each use case. Therefore, the extensive set of benchmarking runs we present should be used as a guidance for further investigations. This said, we have enough data to provide some rough guidance.

If only confidentiality is needed, L1 schemes can perform very efficiently, and we recommend the use of a lightweight, high-security encryption primitive (e.g., QARMA) in a direct mode. If integrity protection is required, but replay attacks are out of scope, L2 schemes with short MACs can be made very efficient by storing the MACs in ECC bits.

Now, let us focus on L3 schemes: nearly-transparent strong memory protection is possible with current technology, but the hardware costs may be prohibitive.

Server SoCs are expensive, with multiple cores and memory channels. Current systems can address a few terabytes of physical memory. The high total system costs allow us to make an argument for counter-based encryption with high arity counter groups stored in on-chip DRAM. The additional cost for counter group storage would be *relatively* minor, such as 1:128 or 1:256 of the external memory. Let us consider a system capable of addressig 4 TiB of physical memory. Placing 64 GiB or more of DRAM in a module close to the main SoC is feasible for client devices today, hence the same technology could be used to place a large tamper-averting memory in a server SoC package, to be used as a large counter group table. When combined with MirE, it would enable the highest level of memory protection at a lower performance impact than all currently deployed schemes without replay protection.

This said, it can be argued that the area budget for such a large memory should rather be used for a system cache, which benefits the *whole* system, and effectively reduces the traffic routed through the MPE. Such a cache could also be dynamically re-partitioned between system and MPE. This re-configuration would rely on an analysis of the traffic and of the impact of the partitioning that goes beyond the scope of this paper.

If these approaches are not possible, storing the integrity tree off-chip and using MirE still provides good performance when combined with a large counter cache.

Regarding MirE, in § 3.3.3.1 we have recalled only two possible configurations for 9-chip DRAM. A taxonomy of configuration with suitable codes for these ad other types of RAS memory, with a security analysis, would help implementors choose the right parameters.

On client devices, RAM usually lacks ECC, making MirE not applicable. However, for use cases such as security modules and business oriented containers, memory bus saturation is less of a concern. We thus expect performance penalties to be contained, usually in line with unloaded systems, and we recommend the use of high arity split counter trees in a dynamically allocated carve-out.

Future work includes contributing our MPE model to the `gem5` project, which we hope will stimulate future research and enable studies for specific workloads and configurations.

Acknowledgments

David Schall’s work was done during two internships at Arm Research and Arm’s Architecture and Technology Group. Part of the work performed during the first internship is documented in his Master’s Thesis [Sch19].

Parts of Ionuț Mihalcea’s work documented in this paper was performed in fulfillment of the requirements for an M.Sc. degree [Mih22]. Ionuț wishes to thank his academic supervisor Prof. Konstantinos Markantonakis, and his line manager at Arm, Paul Howard, for their steady support.

The authors wish to thank Matthias Boettcher, Mike Campbell, Siddhartha Chhabra, Yuval Elad, Wendy Elsasser, Charles Garcia-Tobin, Alexander Klimov, Kazuhiko Mine-matsu, Jason Parker, Prakash Ramrakhiani, Gururaj Saileshwar, Andrew Swaine, Peter Williams, and Nicholas Wood for many useful discussions.

References

- [ABD⁺23] Roberto Avanzi, Subhadeep Banik, Orr Dunkelman, Maria Eichlseder, Shibam Ghosh, Marcel Nageler, and Francesco Regazzoni. The QARMAv2 family of tweakable block ciphers. *IACR Transactions on Symmetric Cryptology*, (3):25–73, Sep. 2023. doi:10.46586/tosc.v2023.i3.25-73.
Cited on page 11.
- [AMD20] AMD. AMD SEV-SNP: Strengthening VM isolation with integrity protection and more, January 2020. Technical Report.
Cited on page 5.
- [AN17] Shaizeen Aga and Satish Narayanasamy. InvisiMem: Smart Memory Defenses for Memory Bus Side Channel. In *Proceedings of ISCA 2017*, pages 94–106, 2017. doi:10.1145/3079856.3080232.
Cited on page 18.
- [ANZ11] Ahmed M. Azab, Peng Ning, and Xiaolan Zhang. SICE: a hardware-level strongly isolated computing environment for x86 multi-core platforms. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *Proceedings of the 18th ACM CCS, 2011*, pages 375–388. ACM, 2011. doi:10.1145/2046707.2046752.
Cited on page 5.
- [App20] Apple Inc. Secure Enclave, 2020. URL: <https://support.apple.com/en-gb/guide/security/sec59b0b31ff/web>.
Cited on pages 5 and 12.
- [Ava17] Roberto Avanzi. The QARMA Block Cipher Family – Almost MDS Matrices over Rings with Zero Divisors, Nearly Symmetric Even-Mansour Constructions with Non-Involutory Central Rounds, and Search Heuristics for Low-Latency S-Boxes. *IACR Trans. on Symmetric Cryptology*, 2017(1):4–44, 2017. doi:10.13154/tosc.v2017.i1.4-44.
Cited on pages 6, 11, 18, and 29.
- [Bac14] Matt Bach. ECC and REG ECC Memory Performance, May 2014. URL: <https://www.pugetsystems.com/labs/articles/ECC-and-REG-ECC-Memory-Performance-560/>.
Cited on page 10.
- [BBB⁺11] Nathan L. Binkert, Bradford M. Beckmann, Gabriel Black, Steven K. Reinhardt, Ali G. Saidu, Arkaprava Basu, Joel Hestness, Derek Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib Bin Altaf, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 Simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, 2011. doi:10.1145/2024716.2024718.
Cited on pages 3 and 18.

- [BCG⁺12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçın. PRINCE — A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract. In *ASIACRYPT 2012*, pages 208–225, 2012. doi:10.1007/978-3-642-34961-4_14. Cited on page 11.
- [BJK⁺16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In *Proceedings of CRYPTO 2016, Part II*, pages 123–153, 2016. doi:10.1007/978-3-662-53008-5_5. Cited on page 11.
- [Blo70] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970. doi:10.1145/362686.362692. Cited on page 16.
- [BLvK18] James Bucek, Klaus-Dieter Lange, and Jóakim von Kistowski. SPEC CPU2017: Next-Generation Compute Benchmark. In *Companion of the 2018 ACM/SPEC ICPE*, pages 41–42. ACM, 2018. doi:10.1145/3185768.3185771. Cited on pages 3 and 18.
- [CL10] David Champagne and Ruby B. Lee. Scalable architectural support for trusted software. In *Proceedings of HPCA 2010*, pages 1–12, 2010. doi:10.1109/HPCA.2010.5416657. Cited on pages 5 and 13.
- [CW79] Larry Carter and Mark N. Wegman. Universal Classes of Hash Functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979. doi:10.1016/0022-0000(79)90044-8. Cited on pages 6, 10, and 12.
- [CXL19] CXL Consortium. Compute express link™ resource library, 2019. URL: <https://www.computeexpresslink.org/resource-library>. Cited on page 18.
- [CZG⁺15] Patrick Colp, Jiawen Zhang, James Gleeson, Sahil Suneja, Eyal de Lara, Himanshu Raj, Stefan Saroiu, and Alec Wolman. Protecting data on smartphones and tablets from memory attacks. In *Proceedings of ASPLOS 2015*, pages 177–189. ACM, 2015. doi:10.1145/2694344.2694380. Cited on page 18.
- [DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2: Lightweight authenticated encryption and hashing. *J. Cryptol.*, 34(3):33, 2021. doi:10.1007/s00145-021-09398-9. Cited on page 11.
- [DR02] Joan Daemen and Vincent Rijmen. AES and the Wide Trail Design Strategy. In *Proceedings of EUROCRYPT 2002*, pages 108–109, 2002. doi:10.1007/3-540-46035-7_7. Cited on page 11.
- [DZWL14] Liang Deng, Qingkai Zeng, Weiguang Wang, and Yao Liu. Equalvisor: Providing memory protection in an untrusted commodity hypervisor. In *13th IEEE TrustCom 2014*, pages 300–309. IEEE Computer Society, 2014. URL: <https://doi.org/10.1109/TrustCom.2014.41>, doi:10.1109/TRUSTCOM.2014.41. Cited on page 5.
- [ECG⁺09] Reouven Elbaz, David Champagne, Catherine H. Gebotys, Ruby B. Lee, Nachiketh R. Potlapally, and Lionel Torres. Hardware mechanisms for memory authentication: A survey of existing techniques and engines. *Trans. Computational Science*, 4:1–22, 2009. doi:10.1007/978-3-642-01004-0_1. Cited on page 3.
- [ECL⁺07] Reouven Elbaz, David Champagne, Ruby B. Lee, Lionel Torres, Gilles Sassatelli, and Pierre Guillemin. TEC-Tree: A Low-Cost, Parallelizable Tree for Efficient Defense

- Against Memory Replay Attacks. In *Proceedings of CHES 2007*, pages 289–302, 2007. doi:10.1007/978-3-540-74735-2_20.
Cited on page 15.
- [EEO⁺14] Dmitry Evtvushkin, Jesse Elwell, Meltem Ozsoy, Dmitry V. Ponomarev, Nael B. Abu-Ghazaleh, and Ryan Riley. Iso-x: A flexible architecture for hardware-managed isolated execution. In *47th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2014, Cambridge, United Kingdom, December 13-17, 2014*, pages 190–202, 2014. doi:10.1109/MICRO.2014.25.
Cited on page 5.
- [FAKM14] Bin Fan, David G. Andersen, Michael Kaminsky, and Michael Mitzenmacher. Cuckoo filter: Practically better than bloom. In Aruna Seneviratne, Christophe Diot, Jim Kurose, Augustin Chaintreau, and Luigi Rizzo, editors, *Proceedings of CoNEXT 2014*, pages 75–88. ACM, 2014. doi:10.1145/2674005.2674994.
Cited on page 16.
- [FLD⁺21] Erhu Feng, Xu Lu, Dong Du, Bicheng Yang, Xueqiang Jiang, Yubin Xia, Binyu Zang, and Haibo Chen. Scalable memory protection in the PENGLAI enclave. In *OSDI 2021*, pages 275–294. USENIX Association, 2021. URL: <https://www.usenix.org/conference/osdi21/presentation/feng>.
Cited on page 5.
- [GJMN16] Robert Granger, Philipp Jovanovic, Bart Mennink, and Samuel Neves. Improved masking for tweakable blockciphers with applications to authenticated encryption. In *Proceedings of EUROCRYPT 2016, Part I*, volume 9665 of LNCS, pages 263–293. Springer, 2016. doi:10.1007/978-3-662-49890-3_11.
Cited on page 12.
- [GMD⁺16] Johannes Götzfried, Tilo Müller, Gabor Drescher, Stefan Nürnberger, and Michael Backes. Ramcrypt: Kernel-based address space encryption for user-mode processes. In *Proceedings of AsiaCCS 2016*, pages 919–924. ACM, 2016. doi:10.1145/2897845.2897924.
Cited on page 18.
- [Gol87] Oded Goldreich. Towards a theory of software protection and simulation by oblivious rams. In *Proceedings of STOC, 1987*, pages 182–194. ACM, 1987. doi:10.1145/28395.28416.
Cited on page 8.
- [GSC⁺03] Blaise Gassend, G. Edward Suh, Dwaine E. Clarke, Marten van Dijk, and Srinivas Devadas. Caches and Hash Trees for Efficient Memory Integrity Verification. In *Proceedings of HPCA’03*, pages 295–306, 2003. doi:10.1109/HPCA.2003.1183547.
Cited on pages 5 and 13.
- [Gue16a] Shay Gueron. A Memory Encryption Engine Suitable for General Purpose Processors. *IACR Cryptol. ePrint Arch.*, 2016. URL: <http://eprint.iacr.org/2016/204>.
Cited on pages 3, 13, 19, and 21.
- [Gue16b] Shay Gueron. Memory Encryption for General-Purpose Processors. *IEEE Secur. Priv.*, 14(6):54–62, 2016. doi:10.1109/MSP.2016.124.
Cited on page 12.
- [HJ05] William Eric Hall and Charanjit S. Jutla. Parallelizable Authentication Trees. In *Proceedings of SAC 2005*, pages 95–109, 2005. doi:10.1007/11693383_7.
Cited on page 13.
- [HJ08] William E. Hall and Charajit S. Jutla. US Patent US US7451310 B2: Parallelizable authentication tree for random access storage, filed Dec. 2, 2002. <http://www.google.com/patents/US7451310>, November 2008.
Cited on page 5.
- [HPL⁺21] Guerny D. H. Hunt, Ramachandra Pai, Michael V. Le, Hani Jamjoom, Sukadev Bhattachipolu, Rick Boivie, Laurent Dufour, Brad Frey, Mohit Kapur, Kenneth A. Goldman, Ryan Grimm, Janani Janakirman, John M. Ludden, Paul Mackerras,

- Cathy May, Elaine R. Palmer, Bharata Bhasker Rao, Lawrence Roy, William A. Starke, Jeff Stuecheli, Enrique Valdez, and Wendel Voigt. Confidential computing for openpower. In Antonio Barbalace, Pramod Bhatotia, Lorenzo Alvisi, and Cristian Cadar, editors, *Proceedings of EuroSys '21*, pages 294–310. ACM, 2021. doi:10.1145/3447786.3456243.
Cited on page 5.
- [HS10] Ruirui C. Huang and G. Edward Suh. IVEC: Off-Chip Memory Integrity Protection for Both Security and Reliability. In *Proceedings of ISCA 2010*, pages 395–406. ACM, 2010. doi:10.1145/1815961.1816015.
Cited on pages 5 and 12.
- [HSH⁺09] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: cold-boot attacks on encryption keys. *Commun. ACM*, 52(5):91–98, 2009. doi:10.1145/1506409.1506429.
Cited on page 9.
- [HT13] Michael Henson and Stephen Taylor. Memory Encryption: A Survey of Existing Techniques. *ACM Comput. Surv.*, 46(4):53:1–53:26, 2013. doi:10.1145/2566673.
Cited on pages 3 and 11.
- [IBM99] IBM. Enhancing IBM Netfinity Server Reliability: IBM Chipkill Memory, 1999.
Cited on page 10.
- [IEE19] IEEE. IEEE standard for cryptographic protection of data on block-oriented storage devices 1619–2018, January 2019. URL: <http://ieeexplore.ieee.org/servlet/opac?punumber=4493431>.
Cited on page 13.
- [IK03] Tetsu Iwata and Kaoru Kurosawa. OMAC: One-Key CBC MAC. In *FSE 2003, Revised Papers*, volume 2887 of *LNCS*, pages 129–153. Springer, 2003. doi:10.1007/978-3-540-39887-5_11.
Cited on page 12.
- [IMO⁺22] Akiko Inoue, Kazuhiko Minematsu, Maya Oda, Rei Ueno, and Naofumi Homma. ELM: A Low-Latency and Scalable Memory Encryption Scheme. *IEEE Trans. Inf. Forensics Secur.*, 17:2628–2643, 2022. doi:10.1109/TIFS.2022.3188146.
Cited on pages 4, 5, 12, 13, and 21.
- [Int21] Intel. Intel[®] Trust Domain Extensions White Paper, August 2021. URL: <https://www.intel.com/content/www/us/en/developer/articles/technical/intel-trust-domain-extensions.html>.
Cited on pages 3, 5, and 24.
- [JACH11] Seongwook Jin, Jeongseob Ahn, Sanghoon Cha, and Jaehyuk Huh. Architectural support for secure virtualization under a vulnerable hypervisor. In *44rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2011, Porto Alegre, Brazil, December 3-7, 2011*, pages 272–283, 2011. doi:10.1145/2155620.2155652.
Cited on page 5.
- [JLK⁺23] Jonas Juffinger, Lukas Lamster, Andreas Kogler, Moritz Lipp, Maria Eichlseder, and Daniel Gruss. CSI:Rowhammer – Cryptographic Security and Integrity against Rowhammer. In *Proceedings of IEEE S&P '23*, 2023.
Cited on pages 5, 10, 12, 18, and 29.
- [JMSS20] Simon Johnson, Raghunandan Makaram, Amy Santoni, and Vinnie Scarlata. Supporting Intel[®] SGX on multi-socket platforms, August 2020. Technical Report.
Cited on pages 3 and 5.
- [JNRV20] Samuel Jaques, Michael Naehrig, Martin Roetteler, and Fernando Virdia. Implementing Grover Oracles for Quantum Key Search on AES and LowMC. In *Proceedings of EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 280–310. Springer, 2020. doi:10.1007/978-3-030-45724-2_10.
Cited on page 17.

- [Kel02] John Kelsey. Compression and Information Leakage of Plaintext. In *Proceedings of FSE 2002*, volume 2365 of *LNCSS*, pages 263–276. Springer, 2002. doi:10.1007/3-540-45661-9_21.
Cited on page 26.
- [KLR⁺20] Mohamed Amine Khelif, Jordane Lorandel, Olivier Romain, Matthieu Regnery, Denis Baheux, and Guillaume Barbu. Toward a hardware Man-in-the-Middle attack on PCIe bus. *Microprocess. Microsystems*, 77, 2020. doi:10.1016/j.micpro.2020.103198.
Cited on page 8.
- [KPW16] David Kaplan, Jeremy Powell, and Tom Woller. AMD Memory Encryption White Paper, April 2016. URL: <https://www.amd.com/system/files/TechDocs/memory-encryption-white-paper.pdf>.
Cited on pages 3 and 5.
- [LAA⁺20] Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, Mohammad Alian, Rico Amslinger, Matteo Androzzi, Adrià Armejach, Nils Asmussen, Srikant Bharadwaj, Gabe Black, Gedare Bloom, Bobby R. Bruce, Daniel Rodrigues Carvalho, Jerónimo Castrillón, Lizhong Chen, Nicolas Derumigny, Stephan Diestelhorst, Wendy Elsasser, Marjan Fariborz, Amin Farmahini Farahani, Pouya Fotouhi, Ryan Gambord, Jayneel Gandhi, Dibakar Gope, Thomas Grass, Bagus Hanindhito, Andreas Hansson, Swapnil Haria, Austin Harris, Timothy Hayes, Adrian Herrera, Matthew Horsnell, Syed Ali Raza Jafri, Radhika Jagtap, Hanhwi Jang, Reiley Jeyapaul, Timothy M. Jones, Matthias Jung, Subash Kannoth, Hamidreza Khaleghzadeh, Yuetsu Kodama, Tushar Krishna, Tommaso Marinelli, Christian Menard, Andrea Mondelli, Tiago Mück, Omar Naji, Krishnendra Nathella, Hoa Nguyen, Nikos Nikoleris, Lena E. Olson, Marc S. Orr, Binh Pham, Pablo Prieto, Trivikram Reddy, Alec Roelke, Mahyar Samani, Andreas Sandberg, Javier Setoain, Boris Shingarov, Matthew D. Sinclair, Tuan Ta, Rahul Thakur, Giacomo Travaglini, Michael Upton, Nilay Vaish, Ilias Vougioukas, Zhengrong Wang, Norbert Wehn, Christian Weis, David A. Wood, Hongil Yoon, and Éder F. Zulian. The gem5 Simulator: Version 20.0+-. *CoRR*, abs/2007.03152, 2020. arXiv:2007.03152, doi:10.48550/arXiv.2007.03152.
Cited on pages 3 and 18.
- [LKS⁺20] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanovic, and Dawn Song. Keystone: an open framework for architecting trusted execution environments. In *Proceedings of EuroSys '20*, pages 38:1–38:16. ACM, 2020. doi:10.1145/3342195.3387532.
Cited on page 5.
- [LMMR21] Gregor Leander, Thorben Moos, Amir Moradi, and Shahram Rasoolzadeh. The SPEEDY family of block ciphers engineering an ultra low-latency cipher from gate level for secure processor architectures. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):510–545, 2021. doi:10.46586/tches.v2021.i4.510-545.
Cited on page 11.
- [MAB⁺13] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. Innovative instructions and software model for isolated execution. In *Proceedings of HASP 2013*, page 10, 2013. doi:10.1145/2487726.2488368.
Cited on page 5.
- [Mer80] Ralph C. Merkle. Protocols for Public Key Cryptosystems. In *Proceedings of the 1980 IEEE S&P*, pages 122–134, 1980. doi:10.1109/SP.1980.10006.
Cited on page 13.
- [Mih22] Ionuț Mihalcea. Prototyping Memory Integrity Tree Algorithms for Internet of Things Devices. Master’s thesis, Information Security Group, Royal Holloway University of London, UK, 2022.
Cited on page 31.
- [MMS⁺20] Tsutomu Matsumoto, Ryo Miyachi, Junichi Sakamoto, Manami Suzuki, Dai Watanabe, and Naoki Yoshida. RAM encryption mechanism without hardware support. *J.*

- Inf. Process.*, 28:473–480, 2020. doi:[10.2197/ipsjjip.28.473](https://doi.org/10.2197/ipsjjip.28.473).
Cited on page 18.
- [MPP⁺08a] Jonathan M. McCune, Bryan Parno, Adrian Perrig, Michael K. Reiter, and Hiroshi Isozaki. Flicker: an execution infrastructure for TCB minimization. In Joseph S. Sventek and Steven Hand, editors, *2008 EuroSys*, pages 315–328. ACM, 2008. doi:[10.1145/1352592.1352625](https://doi.org/10.1145/1352592.1352625).
Cited on page 5.
- [MPP⁺08b] Jonathan M. McCune, Bryan Parno, Adrian Perrig, Michael K. Reiter, and Arvind Seshadri. How low can you go?: recommendations for hardware-supported minimal TCB code execution. In Susan J. Eggers and James R. Larus, editors, *Proceedings of the 13th ASPLOS*, pages 14–25. ACM, 2008. doi:[10.1145/1346281.1346285](https://doi.org/10.1145/1346281.1346285).
Cited on page 5.
- [MPS⁺21] Dominic P. Mulligan, Gustavo Petri, Nick Spinale, Gareth Stockwell, and Hugo J. M. Vincent. Confidential Computing - a brave new world. In *Proceedings of SEED 2021*, pages 132–138. IEEE, 2021. doi:[10.1109/SEED51797.2021.00025](https://doi.org/10.1109/SEED51797.2021.00025).
Cited on pages 3, 5, and 8.
- [MZLS18] Saeid Mofrad, Fengwei Zhang, Shiyong Lu, and Weidong Shi. A comparison study of intel SGX and AMD memory encryption technology. In *Proceedings of the 7th International HASP@ISCA 2018 Workshop, Los Angeles, CA, USA, June 02-02, 2018*, pages 9:1–9:8. ACM, 2018. doi:[10.1145/3214292.3214301](https://doi.org/10.1145/3214292.3214301).
Cited on page 4.
- [NHSQ12] Yao Nianmin, Ma Haifeng, Cai Shaobin, and Han Qilong. A memory integrity protection method based on Bloom filter. In *11th International Symposium on Distributed Computing and Applications to Business, Engineering and Science*, pages 305–309, 2012. doi:[10.1109/DCABES.2012.101](https://doi.org/10.1109/DCABES.2012.101).
Cited on page 17.
- [NIS12] NIST. FIPS PUB 180-4 – Secure Hash Standard. Technical report, National Institute of Standards and Technology, Gaithersburg, MD, United States, March 2012. URL: <https://csrc.nist.gov/publications/detail/fips/180/4/final>.
Cited on page 12.
- [NIS15] NIST. FIPS PUB 202 – SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. Technical report, National Institute of Standards and Technology, Gaithersburg, MD, United States, August 2015. URL: <https://csrc.nist.gov/publications/detail/fips/202/final>.
Cited on page 12.
- [Pet10] Peter A. H. Peterson. Cryptkeeper: Improving security with encrypted RAM. In *Proceedings of IEEE HST 2010*, pages 120–126, 2010. doi:[10.1109/THS.2010.5655081](https://doi.org/10.1109/THS.2010.5655081).
Cited on page 18.
- [PPR05] Anna Pagh, Rasmus Pagh, and S. Srinivasa Rao. An optimal bloom filter replacement. In *Proceedings of SODA 2005*, pages 823–829. SIAM, 2005. URL: <http://dl.acm.org/citation.cfm?id=1070432.1070548>.
Cited on page 16.
- [RCPS07] Brian Rogers, Siddhartha Chhabra, Milos Prvulovic, and Yan Solihin. Using Address Independent Seed Encryption and Bonsai Merkle Trees to Make Secure Processors OS- and Performance-Friendly. In *Proceedings of MICRO-40, 2007*, pages 183–196, 2007. doi:[10.1109/MICRO.2007.44](https://doi.org/10.1109/MICRO.2007.44).
Cited on pages 5 and 13.
- [Rog04] Phillip Rogaway. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In *ASIACRYPT 2004, Proceedings*, pages 16–31, 2004. doi:[10.1007/978-3-540-30539-2_2](https://doi.org/10.1007/978-3-540-30539-2_2).
Cited on pages 6, 10, 12, and 13.
- [SA21] Omais Shafi and Ismi Abidi. CuckoOnsai: An efficient memory authentication using amalgam of cuckoo filters and integrity trees. In *58th ACM/IEEE Design*

- Automation Conference, DAC 2021, San Francisco, CA, USA, December 5-9, 2021*, pages 1273–1278. IEEE, 2021. doi:10.1109/DAC18074.2021.9586205.
Cited on page 17.
- [San14] Andreas Sandberg. *Understanding Multicore Performance: Efficient Memory System Modeling and Simulation*. PhD thesis, Uppsala University, Disciplinary Domain of Science and Technology, Mathematics and Computer Science, Department of Information Technology, Division of Computer Systems, Uppsala, Sweden, 2014.
Cited on page 19.
- [SBS⁺21] Martin Schwarzl, Pietro Borrello, Gururaj Saileshwar, Hanna Müller, Michael Schwarz, and Daniel Gruss. Practical Timing Side Channel Attacks on Memory Compression. *CoRR*, abs/2111.08404, 2021. arXiv:2111.08404, doi:10.48550/arXiv.2111.08404.
Cited on page 26.
- [SCG⁺03a] G. Edward Suh, Dwaine E. Clarke, Blaise Gassend, Marten van Dijk, and Srinivas Devadas. AEGIS: architecture for tamper-evident and tamper-resistant processing. In *Proceedings of ICS 2003*, pages 160–171, 2003. doi:10.1145/782814.782838.
Cited on page 5.
- [SCG⁺03b] G. Edward Suh, Dwaine E. Clarke, Blaise Gassend, Marten van Dijk, and Srinivas Devadas. Efficient Memory Integrity Verification and Encryption for Secure Processors. In *Proceedings of the 36th Annual International Symposium on Microarchitecture*, pages 339–350, 2003. doi:10.1109/MICRO.2003.1253207.
Cited on pages 5 and 15.
- [Sch19] David H. Schall. Evaluation and Optimization of Memory Encryption and Integrity Protection. Master’s thesis, University of Kaiserslautern, Department of Electrical Engineering and Information Technology, Microelectronic Systems Design Research Group, 2019.
Cited on page 31.
- [Shw15] Shweta Shinde and Shruti Tople and Deepak Kathayat and Prateek Saxena. PodArch: Protecting Legacy Applications with a Purely Hardware TCB. Technical Report NUS-SL-TR-15-01, School of Computing, National University of Singapore, February 2015.
Cited on pages 4 and 5.
- [Sko17] Sergei Skorobogatov. How microprobing can attack encrypted memory. In *Proceedings of DSD 2017*, pages 244–251. IEEE Computer Society, 2017. doi:10.1109/DSD.2017.69.
Cited on page 8.
- [SL12] Jakub Szefer and Ruby B. Lee. Architectural support for hypervisor-secure virtualization. In *Proceedings of the 17th ASPLOS, 2012*, pages 437–450. ACM, 2012. doi:10.1145/2150976.2151022.
Cited on page 5.
- [SMS⁺22] Moritz Schneider, Ramya Jayaram Masti, Shweta Shinde, Srdjan Capkun, and Ronald Perez. Sok: Hardware-supported trusted execution environments. *CoRR*, abs/2205.12742, 2022. arXiv:2205.12742, doi:10.48550/arXiv.2205.12742.
Cited on pages 4 and 6.
- [SNOT21] Kuniyasu Suzuki, Kenta Nakajima, Tsukasa Oi, and Akira Tsukamoto. TS-Perf: General Performance Measurement of Trusted Execution Environment and Rich Execution Environment on Intel SGX, Arm TrustZone, and RISC-V Keystone. *IEEE Access*, 9:133520–133530, 2021. doi:10.1109/ACCESS.2021.3112202.
Cited on page 4.
- [SNR⁺18a] Gururaj Saileshwar, Prashant J. Nair, Prakash Ramrakhiani, Wendy Elsasser, José A. Joao, and Moinuddin K. Qureshi. Morphable Counters: Enabling Compact Integrity Trees For Low-Overhead Secure Memories. In *Proceedings of the 50th IEEE/ACM MICRO, 2018*, pages 416–427. IEEE Computer Society, 2018. doi:10.1109/MICRO.2018.00041.
Cited on pages 4, 5, and 15.

- [SNR⁺18b] Gururaj Saileshwar, Prashant J. Nair, Prakash Ramrakhiani, Wendy Elsasser, and Moinuddin K. Qureshi. SYNERGY: Rethinking Secure-Memory Design for Error-Correcting Memories. In *Proceedings of HPCA 2018*, pages 454–465. IEEE Computer Society, 2018. doi:10.1109/HPCA.2018.00046. Cited on pages 4, 5, and 12.
- [SPE18] SPEC. SPEC CPU2017 and simulation research by University of Texas at Austin, 2018. URL: <https://www.spec.org/cpu2017/research/simpoint.html>. Cited on page 19.
- [SPHC02] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. Automatically Characterizing Large Scale Program Behavior. In *ACM SIGPLAN Notices, vol. 37 (Proceedings of ASPLOS-X, 2002)*, pages 45–57. ACM Press, 2002. doi:10.1145/605397.605403. Cited on page 19.
- [TBC⁺20] Meysam Taassori, Rajeev Balasubramonian, Siddhartha Chhabra, Alaa R. Alameldeen, Manjula Peddireddy, Rajat Agarwal, and Ryan Stutsman. Compact leakage-free support for integrity and reliability. In *Proceedings of the 47th ISCA*, pages 735–748. IEEE, 2020. doi:10.1109/ISCA45697.2020.00066. Cited on pages 4 and 16.
- [TJ09] Randy Torrance and Dick James. The State-of-the-Art in IC Reverse Engineering. In *Proceedings of CHES 2009*, volume 5747 of *LNCIS*, pages 363–381. Springer, 2009. doi:10.1007/978-3-642-04138-9_26. Cited on page 9.
- [TSB18] Meysam Taassori, Ali Shafiee, and Rajeev Balasubramonian. VAULT: Reducing Paging Overheads in SGX with Efficient Integrity Verification Structures. In *Proceedings of ASPLOS 2018*, pages 665–678. ACM, 2018. doi:10.1145/3173162.3177155. Cited on pages 4, 5, and 26.
- [UHM⁺20] Rei Ueno, Naofumi Homma, Sumio Morioka, Noriyuki Miura, Kohei Matsuda, Makoto Nagata, Shivam Bhasin, Yves Mathieu, Tarik Graba, and Jean-Luc Danger. High Throughput/Gate AES Hardware Architectures Based on Datapath Compression. *IEEE Trans. Computers*, 69(4):534–548, 2020. doi:10.1109/TC.2019.2957355. Cited on page 29.
- [WB11] Peter Williams and Rick Boivie. CPU support for secure executables. In *Trust and Trustworthy Computing - 4th International Conference, TRUST 2011. Proceedings*, pages 172–187, 2011. doi:10.1007/978-3-642-21599-5_13. Cited on page 5.
- [WWB⁺19] Samuel Weiser, Mario Werner, Ferdinand Brasser, Maja Malenko, Stefan Mangard, and Ahmad-Reza Sadeghi. TIMBER-V: tag-isolated memory bringing fine-grained enclaves to RISC-V. In *Proceedings of NDSS 2019*. The Internet Society, 2019. URL: https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_10-3_Weiser_paper.pdf. Cited on page 5.
- [XLC13] Yubin Xia, Yutao Liu, and Haibo Chen. Architecture support for guest-transparent VM protection from untrusted hypervisor and physical attacks. In *Proceeding of HPCA 2013*, pages 246–257. IEEE Computer Society, 2013. doi:10.1109/HPCA.2013.6522323. Cited on page 5.
- [YA18] Salessawi Ferede Yitbarek and Todd M. Austin. Reducing the Overhead of Authenticated Memory Encryption Using Delta Encoding and ECC Memory. In *Proceedings of DAC 2018*, pages 1–35. ACM, 2018. doi:10.1145/3195970.3196102. Cited on pages 5, 10, 14, and 24.
- [YEP⁺06] Chenyu Yan, Daniel Engleder, Milos Prvulovic, Brian Rogers, and Yan Solihin. Improving Cost, Performance, and Security of Memory Encryption and Authentication. In *Proceedings of ISCA 2006*, pages 179–190, 2006. doi:10.1109/ISCA.2006.22. Cited on pages 5 and 14.